University of Zurich[UZH]

Department of Informatics

Fostering Software Developer Productivity through Awareness Increase and Goal-Setting

Dissertation submitted to the Faculty of Business,
Economics and Informatics
of the University of Zurich

to obtain the degree of
Doktor der Wissenschaften, Dr. sc.
(corresponds to Doctor of Science, PhD)

presented by
André N. Meyer
from Blumenstein, BE, Switzerland

approved in July 2019

at the request of
Prof. Thomas Fritz, Ph.D.
Prof. Margaret-Anne Storey, Ph.D.
Prof. Harald Gall, Ph.D.

University of Zurich[UZH]

The Faculty of Business, Economics and Informatics of the University of Zurich hereby authorizes the printing of this dissertation, without indicating an opinion of the views expressed in the work.

Zurich, July 17, 2019

Chairman of the Doctoral Board: Prof. Thomas Fritz, Ph.D.

Acknowledgments

First and foremost, I would like to thank my great advisor and mentor Thomas Fritz, who supported me throughout my studies and gifted me with the opportunity to pursue my doctoral studies. I very much appreciate the many opportunities he provided me with, to work on amazing research projects, his introductions to researchers that opened the door for some fantastic years-long collaborations, and him constantly motivating (and challenging) me to push myself further, to consider new perspectives and to find simpler solutions to complicated things. Although we were not always living and working on the same continent, he *always* found the time to offer precious advice, feedback and support.

I also want to thank Tom Zimmermann for inviting me several times to work with him at Microsoft, everything he taught me, and all the fun times we had besides doing research, be it by exploring new burger places or at the movies. I thank Gail Murphy for her precious advice and support during my PhD, helping me with participant recruitment, and always making time for a discussion regardless of her full calendar. I thank Peggy Storey for spending her valuable time to examine my thesis and giving precious feedback, for serving on my PhD committee, and for all the valuable discussions we had throughout my PhD. I thank Harald Gall for giving me the opportunity to be part of the SEAL group, serving on my PhD committee, and the interesting discussions we had.

During the time of my PhD, I had the great pleasure to collaborate with researchers from academia and industry, including Christian Bird, Nachi Nagappan, Andy Begel, Rob DeLine, Mary Czerwinski, Daniel McDuff and Spencer Buja

from Microsoft; David Shepherd, Boyang Li, Vinay Augustine, Patrick Francis, Nicholas Kraft and Will Snipes from ABB Corporate Research; Earl Barr from the University College London; Reid Holmes, Nick Bradley, Chris Satterfield, Jan Pilzer from the University of British Columbia; and Chat Wacharamonatham, Alberto Bacchelli, Elaine Huang, Manuela Züger, Sebastian Müller, Katja Kevic, Helen Ai He, Christian Remy and Chia-Kai Yuang from the University of Zurich. I thank them for the interesting projects, valuable discussions, and everything they taught me.

My research would not have been possible without the almost 7'000 study participants and their employers. Thank you for your trust, interest and taking the time to participate in our studies.

Special thanks go to the current and past members of the Software Evolution and Architecture Lab, who made the time of my PhD fun and memorable: Carol Alexandru, Jürgen Cito, Adelina Ciurumelea, Giovanni Grano, Katja Kevic, Christoph Laaber, Philipp Leitner, Sebastian Müller, Sebastiano Panichella, Sebastian Proksch, Gerald Schermann, and Carmine Vassallo. Thank you for your support and valuable feedback on my work, being my guinea pigs for study test-runs, for the restorative Exploding Kittens game-breaks, and the fun times we had at the office, at conferences and retreats. I also thank our university's administration and technicians for always helping us when we need support.

Last but not least, I thank my fiancée, Larissa, for her boundless and unconditional support and encouragement, her understanding when I was abroad to run studies or attend conferences or worked late or on weekends, and joining me on many of my research stays. I thank my family, Dida, Peter and Stephanie for laying a fantastic foundation to work systematically and successfully, always believing in me, and supporting my drive to constantly learn more and grow. Finally, I want to thank Yves, Daniel and Philipp for their precious friendship, their kind support and providing me with positive energy and motivation throughout my PhD.

Abstract

Software development organizations strive to enhance the productivity of their developers. All too often, efforts aimed at improving developer productivity are undertaken without knowledge about how developers spend their time at work and how it influences their own productivity. In our research, we focus on two aspects for improving developers' productivity: better understanding developer productivity and using these findings to foster productivity at work.

To better understand developer productivity, we took a bottom-up approach by investigating developers' perceptions of productivity in the field, and by examining the individual differences in each developer's work. We found that developers spend their time on a wide variety of activities and tasks that they regularly switch between, resulting in highly fragmented work. Our findings further showed that while productivity is very personal, there are observable patterns across developers' productivity perceptions.

Extending our understanding of developers' work and the factors that impact their productivity then allowed us to develop models of developers' work and productivity, and build approaches that support developers with productive behavior changes. To support the identification of self-improvement opportunities that motivate productive behavior changes, we studied how we can increase developers' awareness about work and productivity by combining our models with three persuasive strategies: self-monitoring, self-reflection, and an external indicator. In several areas of life, especially the health and physical activity domains, studies have demonstrated that self-monitoring and self-reflection can increase a user's awareness, and can motivate positive behavior change.

In our work, we investigated how we can map the success of these approaches to software developers' work, by examining their expectations of and experience with self-monitoring tools in the workplace. We implemented the findings as a technology probe into *PersonalAnalytics*, a workplace self-monitoring tool that collects a broad variety of computer interaction data and summarizes the data in a daily and weekly retrospection. A multi-week field-study showed that *PersonalAnalytics* offered meaningful insights to 82% of the participants, but the insights were not actionable enough to motivate behavior change for 41% of our participants. In a follow-up study, we found that continuous and purposeful self-reflection can be an important step towards productive self-improvements in the workplace, since 83% of our participants stated that it supported the identification of goals and actionable strategies, and 80% reported productivity increasing behavior changes. We further studied how we can increase developers' awareness about their co-workers' availability for interruptions, by sensing and externally indicating interruptibility to developers based on their computer interaction. Our large-scale field study with the *FlowLight* showed that we can effectively reduce 46% of external interruptions, participants felt more productive, and 86% of them remained active users even after the two-month study period ended.

Overall, our research showed that we can successfully foster productivity at developers' work, by increasing their awareness about productive and unproductive work habits, and by encouraging work habit improvements based on the gained insights. In addition, our research can be extended and opens new opportunities to foster productive work for development teams. For example, we could aim to gain a better understanding of the interplay between individual and team productivity, to coordinate team collaboration better and improve the scheduling of meetings to be less intrusive.

Zusammenfassung

Softwareentwicklungsunternehmen sind bestrebt, die Produktivität ihrer Softwareentwickler zu steigern. Allerdings werden oft Bemühungen zur Verbesserung der Entwicklerproduktivität unternommen, ohne zu wissen, wie Entwickler ihre Arbeitszeit organisieren und nutzen, und wie ihre eigene Produktivität dadurch beeinflusst wird. Unsere Forschung konzentriert sich auf zwei Aspekte zur Verbesserung der Produktivität von Entwicklern: ein besseres Verständnis der Entwicklerproduktivität und die Nutzung der Erkenntnisse zur Verbesserung der Produktivität am Arbeitsplatz.

Um Einflussfaktoren auf die Entwicklerproduktivität besser zu verstehen, haben wir Produktivität aus der Entwicklerperspektive untersucht. Dafür haben wir die eigene Produktivitätswahrnehmung und individuelle Unterschiede in der Arbeit von Entwicklern untersucht. Unsere Resultate zeigten, dass Entwickler ihre Arbeit mit einer Vielzahl von Aktivitäten und Aufgaben verbringen und regelmässig zwischen ihnen wechseln, was ihre Arbeit in kleine Fragmente aufsplittet. Unsere Ergebnisse zeigten ferner, dass Produktivität zwar sehr individuell ist, es aber beobachtbare Muster in der Produktivitätswahrnehmung von Entwicklern gibt.

Ein besseres Verständnis über die Entwicklerarbeit und die Faktoren, welche Produktivität beeinflussen, erlaubte es uns daraufhin, Modelle zu der Arbeit und Produktivität von Entwicklern zu bauen, um sie bei produktiven Verhaltensänderungen zu unterstützen. Um die Identifizierung der Möglichkeiten von produktiven Verhaltensänderungen zu verbessern, haben wir untersucht, wie wir die Wahrnehmung von Entwicklern zu ihrer eigenen Arbeit und Produk-

tivität erhöhen können. Dafür haben wir unsere Modelle mit drei *Persuasive Strategies* kombiniert: Selbstüberwachung (Self-Monitoring), Selbstreflektion (Self-Reflection), und einem externen Indikator. Studien in verschiedenen Lebensbereichen, insbesondere im Gesundheitswesen und im Sport, haben gezeigt, dass Selbstbeobachtung und Selbstreflexion die Wahrnehmung eines Nutzers in diesen Bereichen erhöhen und positive Verhaltensänderungen motivieren und auslösen können.

In unserer Arbeit haben wir untersucht, wie wir diese erfolgreichen Ansätze auf die Arbeit von Softwareentwicklern übertragen können. Hierfür haben wir sie zu ihren Erwartungen an und Erfahrungen mit bestehenden Selbstbeobachtungssystemen am Arbeitsplatz befragt. Die daraus resultierenden Erkenntnisse haben wir als Technologiestudie in *PersonalAnalytics* integriert, einem System zur Selbstbeobachtung am Arbeitsplatz. *PersonalAnalytics* sammelt eine Vielzahl von Daten zur Interaktion mit dem Computer und fasst die Daten in einer täglichen und wöchentlichen Retrospektive zusammen. Eine mehrwöchige Feldstudie hat gezeigt, dass *PersonalAnalytics* für 82% der Teilnehmer wertvolle Einblicke in die eigene Arbeit und Produktivität bietet. Allerdings waren diese Einblicke für 41% der Teilnehmer nicht konkret genug, um daraus Verhaltensänderungen zu motivieren. Eine Folgestudie hat gezeigt, dass eine kontinuierliche und zielgerichtete Selbstreflektion ein wichtiger Schritt zu produktiven Verhaltensänderungen am Arbeitsplatz sein kann: 83% unserer Teilnehmer haben mit der zielgerichteten Selbstreflektion Ziele und konkrete Strategien zur Selbstverbesserung identifiziert. 80% der Teilnehmer konnten diese daraufhin erfolgreich in produktivitätssteigernde Verhaltensänderungen umsetzen.

Schlussendlich haben wir untersucht, wie wir die Wahrnehmung von Entwicklern über die Kosten von Unterbrechungen bei Arbeitskollegen erhöhen können. Dazu haben wir die Unterbrechbarkeit von Entwicklern basierend auf ihren Computerinteraktionen gemessen und mit einem externen Indikatoren angezeigt. Eine grosse Feldstudie mit *FlowLight* hat gezeigt, dass die Anzahl der persönlichen Unterbrechungen nach der Einführung von *FlowLight* um 46% gesunken ist. Ausserdem fühlten sich die Teilnehmer produktiver. 86% der Teilnehmer nutzten *FlowLight* auch nach Ablauf der zweimonatigen Studienzeit weiterhin aktiv.

Zusammenfassend zeigten unsere Erkenntnisse, dass sich die Produktivität von Softwareentwicklern erfolgreich steigern lässt, indem man ihr Bewusstsein für produktive und unproduktive Arbeitsgewohnheiten erhöht und sie ermutigt, ihre eigenen Arbeitsgewohnheiten basierend auf den gewonnenen Erkenntnissen zu verbessern. Darüber hinaus zeigen wir auf, wie unsere Forschung erweitert werden kann, und welche neuen Möglichkeiten sich eröffnen, um die Produktivität von Entwicklerteams zu erhöhen. Beispielsweise könnten wir ein besseres Verständnis über das Zusammenspiel von individueller Produktivität und Team-Produktivität anstreben, das zu einer verbesserten Koordination von Teamarbeit oder der Planung von Meetings während weniger störenden Zeiten beiträgt.

Contents

List of Figures

List of Tables

Acronyms and Definitions

Activity is an action undertaken by the developer during his or her work, *e.g.*, navigating code or reading an email.

API application programming interface

Context or task context, refers to the mental model the developer builds at work; consisting of information, resources and artifacts about one or several tasks.

DnD do not disturb

Goal or work habit goal, describes a target or outcome habit that developers set for themselves to improve their productivity.

h hour(s)

IDE integrated development environment

IM instant messaging

Interruptibility is a person's availability for interruptions.

min minute(s) or minimum

max maximum

PI is short for personal informatics, and describes tools and methods to self-quantify oneself and identify opportunities for self-improvements. Other related terms are self-tracking, self-monitoring, and self-surveilling.

RQ research question

s second(s)

Self-Monitoring refers to the process where a user *continuously* tracks certain behaviors, either automatically using a self-monitoring tool or manually.

Self-Reflection refers to the process where a user *purposefully* reviews and thinks about his or her behaviors.

Stdev or std. dev. standard deviation

Strategy is the system developers employ to make progress towards and eventually reach their (work habit) goals.

Task is a work assignment with a specific objective that developers divide their work into, *e.g.*, fixing a bug or preparing for a team meeting.

TTM is short for Transtheoretical Model of behavior change, invented by Prochaska and Velicer [1997]. TTM models behavior change as a sequence of stages which the person advances through until a behavior change can be maintained.

1

Synopsis

There is a common refrain that repeats itself in similar forms every couple of years: our inability to produce enough software to satisfy the needs of the world. For example, in 1968, attendees at the first NATO software engineering conference coined the term software crisis [Naur and Randell, 1969] and Andreessen [2011] wrote about software eating the world, expressing that the need for software keeps outstripping our ability to produce it.

There are a couple of ways of addressing the gap between software demand and supply. We could try to reduce the demand, which seems unlikely to succeed. Or, we could try to increase the supply, namely our ability to produce software. Our research targets the latter, specifically to increase the supply of software by improving software developers' productivity. In particular, we focus on two aspects for improving developers' productivity: better understanding developer productivity and using these findings to foster productivity at work.

Examining Productivity. A substantial amount of research on developer productivity has been undertaken over the past four decades. The majority of this research focused on defining and measuring productivity from a *top-down perspective*, formally defined as the ratio between output and input in the Oxford-Dictionary [2019]. In particular, this research aimed at quantifying productivity solely in terms of output measures, *i.e.*, the artifacts and code created per unit of time, for example the lines of source code modified per hour [Devanbu et al., 1996], the number of function points created per day [Jones, 1994], or the tasks completed per month [Zhou and Mockus, 2010]. Another body of research considered *technical* factors associated with productivity, such as software size [Albrecht, 1979] and complexity [Brooks Jr, 1995], and *soft* factors, such as the effect of workplace settings, meetings and interruptions, with the latter ones shown to be one of the biggest impediments to productivity (*e.g.*, [Czerwinski et al., 2004; DeMarco and Lister, 1985; Mark et al., 2005; Parnin and Rugaber, 2011; Perry et al., 1994b]). However, these studies generally do not consider the specifics of developers' workdays, and the individual differences in developers' work that might affect productivity [Gonçalves et al., 2011; Singer et al., 2010]. As a result, only very little is known about how developers themselves perceive their own productivity and how these insights could be used to foster productive work [Rooksby et al., 2016; Treude and Storey, 2010]. In our research, we aimed to get a deeper understanding of software developers' work from a *bottom-up perspective*, *i.e.*, the individual developer perspective, by investigating when developers perceive themselves as productive, how they organize their workdays, what factors impact their productivity, and how their perceived productivity relates to their work activities.

Fostering Productivity. An improved understanding of the factors that impact developers' work and productivity allows to better support developers at their work. However, most developers are not aware of how these factors impact their own productivity, even though they are generally interested in better understanding and improving themselves [Li et al., 2015; Perry et al., 1994b]. In a second step of our research, we aimed to increase developers' awareness about

their productive and unproductive habits and to motivate self-improvements, by supporting them through three persuasive technologies: self-monitoring, self-reflection, and an external indicator. Persuasive technology, as coined by Fogg [2003], refers to technology that is designed to change users' attitudes or behaviors through persuasion and social influence, but not coercion. In several areas of life, especially the health and physical activity domains, these persuasive technologies have been shown to provide users with meaningful insights into their behavior, which motivate positive behavior changes, such as becoming more active or losing weight (*e.g.*, [Consolvo et al., 2008b; Fogg, 2003; Gasser et al., 2006; Kersten-van Dijk et al., 2017; Lee et al., 2017; Munson and Consolvo, 2012]). More recently, researchers have looked into using *automation* to assist with workplace self-monitoring, by logging users' computer interaction data. Approaches such as RescueTime [2019], TimeAware [Kim et al., 2016] and Codealike [2019] have largely focused on top-down approaches using pre-defined measures of productivity that are visualized to the user. In software development, however, little is known about developers' expectations of, requirements for, and impact of these approaches on their behavior [Rooksby et al., 2014; Treude and Storey, 2010]. The Personal Software Process (PSP) by Humphrey [1996] has taken a first step towards workplace self-monitoring and -reflection for developers. PSP allows the monitoring of and reflecting on a set of basic metrics, such as time estimates and quality. While the method showed great potential to improve developers' performance and quality of software, applying it in practice was shown to be time consuming and error prone, since PSP requires to collect the data *manually* [Johnson and Disney, 1998]. In our work, we aimed to extend PSP by developing and evaluating a self-monitoring approach that combines self-reports with *automatically* collected data. We used our insights on developers' work and productivity, and the automatically captured computer interaction data, to develop models. Finally, we explored how to best communicate these models as insights to developers to motivate productive self-improvements at work, for example through dashboards within the self-monitoring system or even through an external indicator.

In summary, we aimed to foster developer productivity at the workplace. To achieve this objective, we first developed models of work and productivity by studying developers work and productivity, and then integrated these models into tool support that we evaluated through various field studies.

This leads to the following ***hypothesis***:

Hypothesis: Models of developers' work and their computer interaction can be used to foster developer productivity through *a)* self-monitoring, *b)* self-reflection, and *c)* an external indicator.

To investigate our hypothesis, we focused on three main research questions that are described in Chapter 1.1. Chapter 1.2 provides an overview of the approach and main findings, consisting of seven studies we performed to answer our research questions. We discuss the main threats to validity of our approach in Chapter 1.3, challenges in Chapter 1.4, potential future work in Chapter 1.5, and related work in Chapter 1.6. Finally, we summarize the contributions of our work in Chapter 1.7 and provide a roadmap of this thesis in Chapter 1.8.

1.1 Research Questions

To validate our hypothesis, we examined the following ***research questions***:

Research Question 1: How does a software developer's workday look like in terms of *a)* activities, *b)* tasks, *c)* work fragmentation, and *d)* perceived productivity?

Based on our improved understanding of developers' work and productivity, we further explore how to foster productivity:

Research Question 2: Can we foster productivity by increasing developers' self-awareness about work and productivity through *a)* self-monitoring and *b)* reflective goal-setting?

Research Question 3: Can we foster productivity by increasing developers' external awareness about work and productivity through modeling and externally indicating interruptibility?

Figure 1.1 gives an overview of the relationship between the research questions.

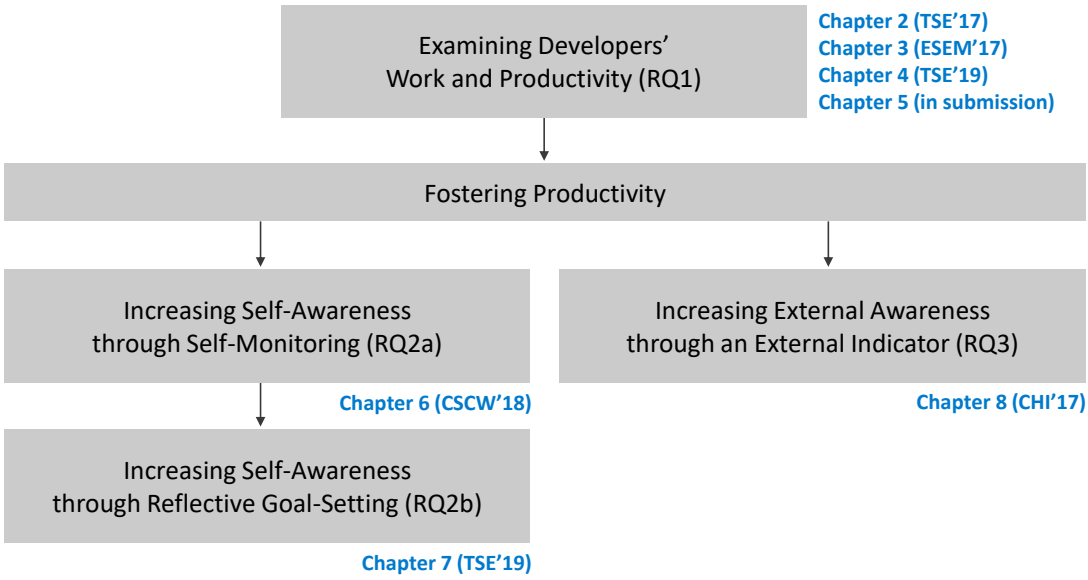
RQ1: Examining Developer Work and Productivity. Several studies have looked into developers' work and productivity. Most have generally focused on specific factors, such as meetings, emails or interruptions, which were shown to impact developers' work, decrease productivity, and negatively impact the quality of output (*e.g.*, [Boehm, 1987; DeMarco and Lister, 1985; Meyer et al., 2014; Murphy-Hill et al., 2019]). Yet, little is known about how developers' structure and organize their workdays, and how developers perceive their productivity throughout these days. In RQ1, we examined developers' workdays more holistically, and investigated individual differences between developers' work. This includes gaining a better understanding of the time they spend at work in different activities and tasks, their habits at work, and how their work influences their perceptions of productivity.

RQ2: Increasing Self-Awareness to Foster Productivity. In other areas of life, especially the physical activity and health domain, self-monitoring and self-reflection have been shown to increase users' awareness about positive and problematic behaviors (*e.g.*, [Bentley et al., 2013; Consolvo et al., 2008a,b; Fritz et al., 2014]). The gained insights then motivated users to define self-improvement goals, which fostered positive behavior changes (*e.g.*, [Gasser et al., 2006; Locke and Latham, 2002; Monkaresi et al., 2013; Munson and Consolvo, 2012]). While several related approaches for the workplace already exist (*e.g.*, [Kim et al., 2016; RescueTime, 2019; Whittaker et al., 2016]), little is known about developers' expectations of and experience with these tools, and how they eventually impact developers' work and productivity. In our research, we want to map the positive impact of self-monitoring and self-reflection to the software engineering domain. With RQ2a, we studied how to leverage self-monitoring to increase developers' self-awareness about work and productivity, by allowing them to self-monitor a broad variety of aspects relevant to their work that we identified in RQ1. In particular, we first examined developers' requirements for a workplace self-monitoring tool. We then evaluated how they are using and engaging with such

a tool in practice, and how the insights should be communicated best to be actionable enough to motivate productive behavior changes. In RQ2b, we ran a follow-up study to explore how purposefully self-reflecting about work and productivity can support the identification of opportunities for self-improvements, and the definition of goals and actionable strategies that motivate productive behavior changes.

RQ3: Increasing External Awareness to Foster Productivity. Previous work has identified interruptions as one of the most prominent factors that can impact developers' productivity. While interruptions can also be beneficial, for instance to resolve problems quickly [Isaacs et al., 1997], they can be very costly when they happen at an in-opportune moment, such as when the developer is very focused or engaged in a difficult task. Amongst other insights on interruptions, studies have found that they occur frequently and interrupt developers' work on their main tasks, which takes more time to finish them and results in more errors (*e.g.*, [Bailey et al., 2001; Czerwinski et al., 2004; González and Mark, 2004; Parnin and Rugaber, 2011]). In our work, we leveraged our findings from RQ1 to develop a model of developers' availability for interruptions (related to their focus), and used it to increase developers' awareness about their co-workers' focus. Subsequently, we studied how to present the model using an external indicator to increase awareness about and discourage in-person interruptions.

Figure 1.1: Overview of our research questions.



1.2 Research Approach and Main Findings

To explore and answer our research questions, we conducted a variety of studies with professional software developers, including field studies, observations and surveys. An overview of our studies is summarized in Table 1.1. In the following, we provide a summary of the main findings and insights from each study. More details and further results are provided in Chapters 2 to 8, as well as in the corresponding publications.

Table 1.1: Overview of the studies we conducted to answer our research questions.

Study	RQ	CI Monitoring	Exp. Sampling	Observation	Interview	Survey	#Part.	Length	Chapter
Workdays & Productivity	1	✓	✓		✓		20	2 weeks	2
Productivity Personas	1					✓	413		3
Good & Typical Workdays	1					✓	5971		4
Task Switches and Types	1	✓	✓	✓		✓	25	4 hours, 2-4 days	5
Self-Monitoring	2	✓	✓			✓	63	3 weeks	6
Reflective Goal-Setting	2		✓			✓	52	2-3 weeks	7
External Indicator	3	✓	✓		✓	✓	449	2 months	8

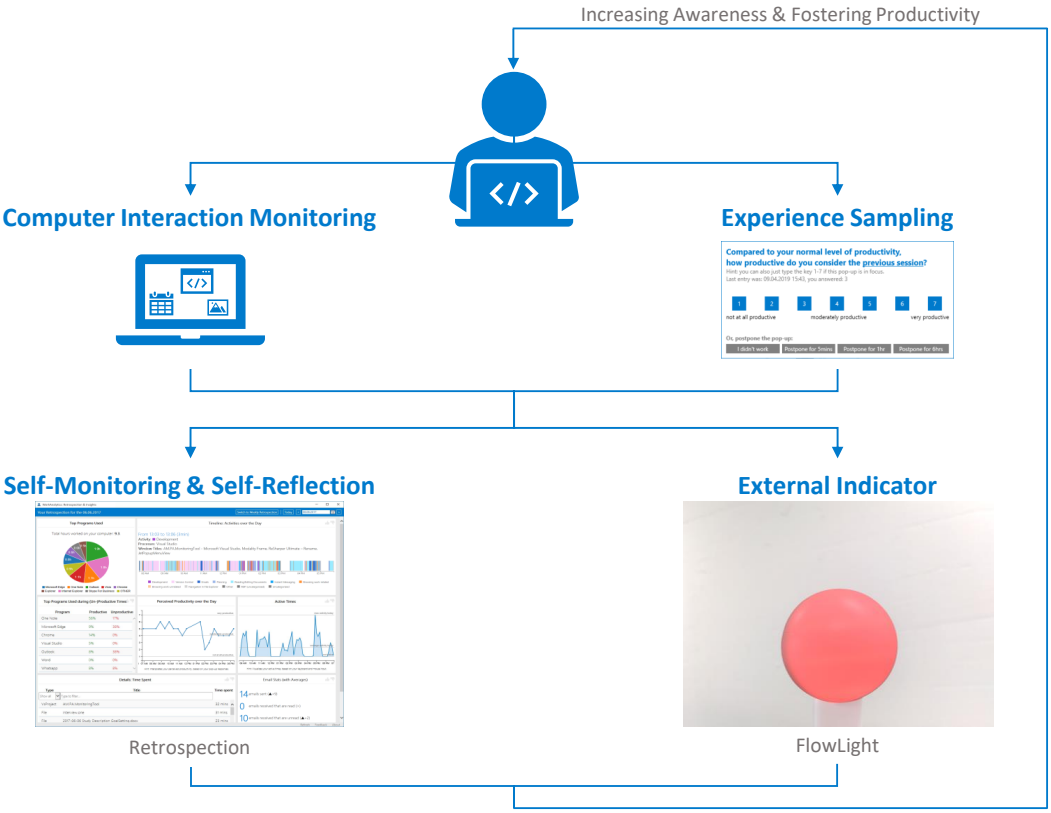
CI: computer interaction, # Part: number of professional software developers studied.

As tool support for four of our studies, we developed and used *PersonalAnalytics*. *PersonalAnalytics* consists of a monitoring component that we built to *automatically* capture developers' computer interaction, including user input, application use, documents and development projects accessed, websites visited, emails/IMs sent/received, meetings scheduled, and code reviews worked on. *PersonalAnalytics* also contains an experience sampling [1] component that allowed us to prompt study participants in periodic intervals to self-report their perceived

[1] Experience Sampling Method (ESM) is a research method to ask participants at periodic intervals to self-report some behavioral or mental aspects [Consolvo and Walker, 2003; Larson and Csikszentmihalyi, 2014].

productivity, task switches, or interruptibility. The collected data was then used to model developers' work and productivity, and visualized in the Retrospection, a daily and weekly summary of developers' work (RQ2). The collected data was further used in the *FlowLight*, to sense and externally indicate a developers' availability for interruptions (RQ3). An overview of the components that provide the basis for *PersonalAnalytics* is presented in Figure 1.2. *PersonalAnalytics* is available on GitHub [2].

Figure 1.2: Overview of the components used in *PersonalAnalytics*.



[2] https://github.com/sealuzh/PersonalAnalytics

1.2.1 RQ1: Examining Developer Work and Productivity

The objective of RQ1 is to increase our understanding of developers' work habits and how they influence productivity. To gain a broader understanding and better generalize the results, we performed a series of four studies using a variety of methods that ranged from observations to multi-week field studies with *PersonalAnalytics* deployed, and studying professional software developers working for seven different companies. Based on our analysis of the gathered data, our main observation is that developers spend their time on a wide variety of activities and tasks that they regularly switch between, resulting in highly fragmented work. Our findings further showed that while productivity is very personal, there are observable patterns across developers' productivity perceptions.

Workdays & Productivity. **[RQ1 a) activities, b) tasks, d) perceived productivity]** In our first study, we aimed at investigating the activities developers pursue at work, what fragments their work, and how their work habits related to their perceptions of productivity. To that purpose, we conducted a two-week field study with 20 professional software developers from various companies. We deployed the monitoring component of *PersonalAnalytics*, and employed experience sampling in the form of periodic self-reports on developers' perceived productivity. With the study, we collected a dataset of 220 workdays from the 20 participants, resulting in a total of 2197 hours of computer usage data and self-reports. To get a higher-level view on developers' work, we developed an automated mapping of the applications that participants used into activities.

The analysis of the data we collected with the monitoring component revealed that developers spend their time on a wide variety of activities and switch regularly between them, which results in highly fragmented work. On average, developers remain in an activity only between 20 seconds and 2.0 minutes. Overall, a typical workday is on average 8-9 hours long, with one fourth of it spent with coding related activities, and another fourth spent with collaborative activities such as meetings, emails and IM. There are, however, notable differences between teams and companies, such as the time spent on emails ranged from less than one minute to more than an hour a day for developers in different companies.

Investigating developers' self-reported productivity ratings revealed large variations amongst developers and that productivity is a highly personal matter. A closer investigation revealed that there are observable patterns across developers' productivity perceptions. For example, we found that developers' productivity perceptions follow the same habitual patterns each day as each can be roughly clustered into either morning-person (20%), low-at-lunch person (35%), and afternoon-person (40%). We also built explanatory models to study which factors contribute most to developers' perceptions of productivity. None of the explanatory models that we built relating actions and activity to perceived productivity was able to explain the productivity perceptions of a large number of developers. However, many developers consider email, planned meetings and work unrelated browsing as less productive activities, and usually perceive themselves as more productive when they have a higher user input and spend their time with coding.

Productivity Personas. **[RQ1d) perceived productivity]** The individual differences in developers' productivity perceptions will make it more challenging to determine meaningful, *universal* behavior change plans to foster productivity for RQ2 and RQ3. To explore whether we can determine further groups of developers with similar perceptions of productivity, and better understand how various productivity factors influence them differently, we conducted an online survey at one large software organization with 413 professional software developers. First, participants rated a list of previously identified productivity factors based on importance, including the coding time, email workload, interruptions and workplace setting. After normalizing participants' responses, we used a clustering algorithm, called Partitioning Around Medoids (PAM) [Kaufman and Rousseeuw, 1987], to identify groups of developers with similar productivity perceptions.

Overall, we identified six personas describing developers' productivity perceptions: social, lone, focused, balanced, leading and goal-oriented developers. *Focused developers*, for example, feel most productive when they are working on a single task at a time, with few interruptions. Contrary, *social developers* feel most productive when they are helping others and collaborate frequently. These

personas are a first step towards a set of *developer productivity traits*, such as goal-orientation, single-task focus, or socialness. These traits can then help to better understand and compare developers' individual productivity perceptions, and to tailor approaches for improving work and productivity to these personas. More details can be found in Chapter 3.

Good & Typical Workdays. **[RQ1a activities]** While our first study revealed early insights into *typical* developer workdays, we were interested in better understanding which factors influence typical and atypical workdays, and what makes them good or bad. We were further interested in investigating similarities in developers' workday structures and in studying how developer workdays are impacted by meetings and interruptions. We conducted a large-scale survey collecting 5971 responses at a multi-national software development organization. In our qualitative analysis of the survey, we applied a coding strategy, consisting of Open Coding, Axial Coding and Selective Coding iterations, as defined by Strauss and Corbin [1998]'s Grounded Theory. We then also quantitatively analyzed the collected data, by comparing the time developers spend on activities on workdays they considered good and typical, clustering their workdays into workday types, and investigating the effect that various collaborative activities have on developers' perceptions of good and typical workdays.

Extensive team discussions of the categories that resulted from the coding strategy allowed us to understand their relationships and develop two conceptual frameworks that characterize developers' good and typical workdays (see Chapters 4.5.1 and 4.5.2). Our investigation into the factors that influence how good and typical developers perceive their workdays highlighted the importance of agency, *i.e.*, developers' ability to control how they organize their work and reduce factors that randomize work, including infrastructure issues, administrative tasks, interruptions or unplanned meetings. While previous work uniformly concludes that meetings and interruptions are unproductive overall, the scale of our survey allowed us to uncover nuances, such as that their impact depends on the development phase: during specification, planning and release phases, they are common, but productive.

Task Switches and Types. [RQ1b) tasks] Since we found that task work and task switches are an important factor for fragmenting developers' work and impacting their productivity, we were further interested in better understanding how developers organize work on the granularity of tasks, and how they switch between tasks. To that purpose, we developed a model to *automatically* detect developers' task switches and task types, based on their computer interaction data and the semantic information of the artifacts they work with. In two field studies, one 4-hour observational study and a multi-day study using experience sampling, we collected task data from 25 professional software developers using the monitoring component of *PersonalAnalytics*. After combining and cleaning the data from the two studies, we extracted a total of 42 temporal and semantic features that we used to train a machine learning classifier and evaluated the resulting models' ability to predict task switches and task types in the field. We built both, individual models, to train and test with data solely from one participant, and general models to train with data from all participants except one and to test on the remaining one.

From the various classification algorithms that we trained and tested; Random Forest resulted in the highest accuracy. Our analysis revealed that it is possible to use temporal and semantic features from developers' computer interaction data to *automatically* detect task switches and task types in the field with high accuracy of 87% and 61% respectively. On average, participants switched tasks 6.0 times per hour and spent 13.2 minutes on a task before switching, thus, confirming the high fragmentation of developers' work that we identified before. Besides improving our understanding of developers' tasks and how switching between them fragments developers' work, our *automated* task detection makes it possible to better support existing approaches that capture task context, reduce task switching and support task resumption, which previously required *manual* interaction of the developer, *e.g.*, by self-reporting the task boundaries (*e.g.*, [Dragunov et al., 2005; Fogarty et al., 2005; Kersten and Murphy, 2006; Züger et al., 2017]).

1.2.2 RQ2: Increasing Self-Awareness to Foster Productivity

We conducted two field studies to answer RQ2. In the first study, we evaluated the potential of workplace self-monitoring to increase developers' awareness about work and productivity (RQ2a). In a follow-up study, we explored how purposefully self-reflecting about work and productivity can support the identification of opportunities for self-improvement, and the definition of goals and actionable strategies that motivate productive behavior changes (RQ2b).

Self-Monitoring (RQ2a). To examine how self-monitoring can increase developers' self-awareness about productivity and work, we followed a mixed-methods approach. First, we used iterative, user-feedback driven development (N=20) and a survey (N=413, survey from above) to infer design elements for workplace self-monitoring. We then included the inferred design elements into *PersonalAnalytics*, by adding a self-monitoring component, and by simplifying our experience sampling component prompting users to self-report productivity to be less intrusive and quicker to respond to. To enable multi-faceted insights in the self-monitoring component, the captured data is aggregated and visualized in a daily retrospection (see Figure 1.3), and a weekly summary. Next, we conducted a field study with 43 professional software developers during three work weeks to evaluate the experience with and the impact of *PersonalAnalytics*. Throughout the study, we collected qualitative and quantitative data from participants through multiple surveys, email feedback, and usage logs of *PersonalAnalytics* that we analyzed again using a coding strategy informed by Strauss and Corbin [1998]'s Grounded Theory approach.

Our analysis showed that the retrospection increases developers' self-awareness about work and productivity. 82% said that the retrospections' visualizations of the personalized list of measures increased their awareness and provided novel insights. For example, developers learned about the time they spent collaborating or making progress on tasks, how their productivity changes over the course of a workday, or the extent of their work fragmentation. The retrospection further helped to rectify misconceptions users had about their work, such as how much time they actually spend with emails or work-unrelated web browsing

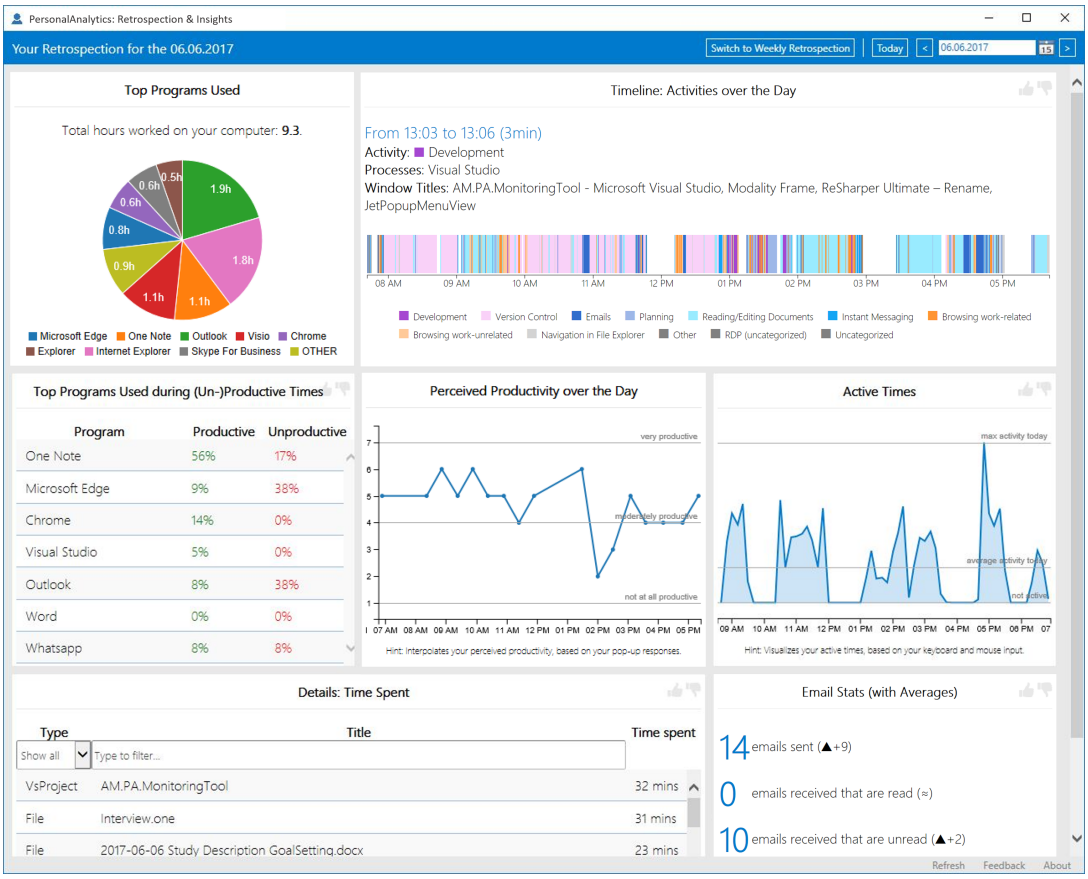
(*e.g.*, on Facebook). While we initially used the experience sampling method to collect data about developers' productivity perceptions, the final questionnaire revealed that the periodic self-reports increased the awareness about work for 59.2% of all participants. *PersonalAnalytics*' ability to let users self-report their productivity instead of automatically measuring it, was highly appreciated, since many participants do not think an automated measure can accurately capture an individual's productivity. It further helped them to assess whether they spent their past work hour efficiently, whether they worked on something of value, and whether they made progress on their current task.

We further investigated if the insights gained from *PersonalAnalytics* can foster productive self-improvements. We learned that slightly less than half of the participants (41%) changed a few of their habits, including trying to better plan their work (6%), trying to optimize how they spend their time with emails (13%), or trying to focus more and avoiding distractions (19%). 41% of our participants reported not having changed any behaviors. Several participants stated that the visualizations and insights were not concrete and actionable enough to know what or how to change, and they suggested to add recommendations to *PersonalAnalytics* that suggest self-improvements at work. These recommendations ranged from pop-ups to recommend a break from work, all the way to intervening or blocking specific applications or websites for a certain time.

Reflective Goal-Setting (RQ2b). To explore how we can provide developers with more actionable insights that allow them to define self-improvement goals and foster productive behavior changes, we ran another field study. Inspired by Humphrey [1996]'s Personal Software Process (PSP) and diary studies in other areas of research, we combined self-reflection with goal-setting, and designed a reflective goal-setting study. In this study, we asked 52 professional software developers to reflect about work on a daily basis for several weeks, as well as define and refine goals and actionable strategies to improve their work habits.

Our reflective goal-setting study resulted in a rich set of work habit goals and strategies that we analyzed. The goals can be broadly categorized into

Figure 1.3: Screenshot of the daily retrospection in *PersonalAnalytics*.



improving time management, avoiding deviations from planned work, improving the impact on the team, maintaining work-life balance, and continuous learning. We found that *continuous* self-reflection can be an important step towards productive self-improvements in the workplace, since participants stated that it supports the identification of goals (81%) and actionable strategies (83%). The daily self-reflections not only increased developers' awareness about work habits, progress and achievements (85%), but also led to productive behavior changes (80%). As a result, while initially being skeptical towards "journaling" their work, most participants (96%) stated afterwards that they could imagine to continue self-reflecting on a regular basis. Few participants, however, mentioned that constantly self-reflecting may increase pressure to always perform well and could,

thus, turn into a burden without tool support that would make self-reporting more convenient. We conclude that *continuous* reflective goal-setting enables developers to improve and maintain good work habits. In Chapter 7, we discuss these results with regards to prior work on self-reflection with other types of knowledge workers, and how tools could support developers with their reflective goal-setting and how they might foster long-term self-reflection.

1.2.3 RQ3: Increasing External Awareness to Foster Productivity

External Indicator. To increase developers' awareness of their co-workers' interruptibility at work and discourage interruptions at in-opportune moments, we developed and studied the *FlowLight*. The *FlowLight* is based on the monitoring component of *PersonalAnalytics* and combines a physical interruptibility indicator in the form of a traffic-light like LED with an automated interruptibility measurement (see Figure 1.4). Our interruptibility is based on a developer's computer interaction and approximates how interruptible (or focused) the developer is. Inspired by traffic-lights, the *FlowLight* changes its color to indicate a user's interruptibility: *available* for interruptions as green, *busy* as red, *do not disturb* as pulsating red, and *away* as yellow. The user's interruptibility state is calculated in real-time, based on mouse and keyboard input [3], the user's historical interaction data, and a smoothing function. Whenever a user's interruptibility state changes, the *FlowLight* updates the color of the LED light. It further changes the user's Skype for Business presence status to additionally reduce interruptions over IM. We evaluated the accuracy of our interruptibility measure and *FlowLight*'s ability to increase the external awareness and reduce in-person interruptions. To that purpose, we conducted a large-scale field study with 449 participants working for one multi-national company in 12 different countries. We asked participants to self-report in-person interruptions during one week before installing the *FlowLight*, and again a week after they familiarized themselves with the *FlowLight* for another week (*i.e.*, to reduce Hawthorne-type or novelty

[3]These data sources were selected since they can be measured non-invasively and to limit privacy concerns.

Figure 1.4: *FlowLights* mounted on study participants' cubicle walls.

(a) *FlowLight* (b) *FlowLights* mounted in an office in India

effects). At the end of the study, we conducted surveys and interviews to learn more about the impact and value of *FlowLight*, which resulted in a rich dataset of 36 interruption logs, 23 interview transcripts, 183 survey responses, 47 *FlowLight* usage data logs, and activity logs from all 449 participants.

Analyzing the 36 interruption logs showed that continuously indicating the interruptibility of co-workers significantly reduces the amount of in-person interruptions by 46%. Participants reported higher productivity as a result of experiencing less costly interruptions, which increased the time they could spend focused on their work tasks. Another insight was that the *FlowLight* increases users' awareness about the cost of interruptions. For example, participants reported that they now think more about whether interrupting a co-worker is necessary, and if not, they try to find a more suitable time. Participants also reported that the *FlowLight* encouraged focusing for longer intervals on their work tasks, especially when it turned red and they wanted to keep it that way. We found that 71% of the participants perceived *FlowLight*'s interruptibility status as accurate. Nonetheless, there is potential for improvement, such as times

they are not actively working on the computer (*e.g.*, doing paper work) or when they are thinking or reading. For example, webcams or biometric sensors worn by the users might help to better understand their activities and might help to increase the accuracy of the interruptibility classifier (*e.g.*, [McDuff et al., 2012; Samara et al., 2017; Züger et al., 2018]). Even two months after the installation, 86% of participants continued using the *FlowLight*. The *FlowLight*'s successful application over multiple months demonstrates that combining an external indicator with an automated interruptibility measurement that is based on computer interaction data is effective to reduce costly in-person interruptions and increase productivity. More details about our findings can be found in Chapter 8.

1.3 Threats to Validity

In the following, we present our main threats to the validity of our research, and discuss how we addressed and mitigated them.

External Validity. A common challenge of field studies, limiting the generalizability of the results, is having access to only a small set of participants and companies over a limited time span. In our studies, we took a number of steps to mitigate threats to generalizability, including studying only professional software developers (no students) with a broad range of experience (0.5 to 40 years), and working for 17 different companies (startups to multi-national corporations) in 12 different countries (including Canada, US, India, Brazil, Korea and Switzerland).

In three studies, the external validity of our evaluation of *PersonalAnalytics* and the *FlowLight* might be threatened by only having participants from a single (multi-national) corporation each. To mitigate the risks, we ran the studies at many separate teams within each corporation, in different subsidiaries of the corporation, working on different projects and with different development processes. Previously, empirical studies performed on a single case (*i.e.*, one company) were shown to contribute to scientific discovery [Flyvbjerg, 2006].

The voluntary participation in our studies may also result in a certain self-selection bias towards participants who are more willing to quantify various

aspects of their life and use the collected data to improve themselves, which is a common threat of studies applying or evaluating personal informatics systems [Kersten-van Dijk et al., 2017]. To reduce this risk, we openly advertised the studies to encourage *any* professional software developer to join, were very transparent and upfront about the study objectives, pro-actively answered privacy concerns, and let participants continue their regular work to minimize the studies' intrusiveness.

Since developers' work and activities vary greatly depending on the current project stage and daily discrepancies, another limitation might be that our field studies only capture a small excerpt of developers' work, especially for the ones we conducted over only a few days. To mitigate these risks, we recruited participants from various companies, working in different teams and at different stages of their project, and we asked them about the representativeness of their work to make sure the studied timeframe was representative of their regular work (*i.e.*, not particularly busy or quiet). We further plan to replicate our studies by considering additional variables, such as company culture, size and processes, as well as developers' personality or gender (Chapter 1.5).

Internal Validity. Running a monitoring application on participants' computers might pose privacy concerns to participants or their companies, *e.g.*, they might feel observed or fear their managers will get access to the data. To reduce these concerns, we were as transparent and explicit as possible about the data we collected, the people who have access, and how we planned to analyze the data. In all studies, participants could access the raw data logs and were given the opportunity to obfuscate them. Additionally, we logged the data only locally on the participants' machines, and never automatically uploaded the data without participants' prior consent and option to manually check the data package before sharing it with us. We further never made the logged data available to participants' co-workers or managers.

While our reliance on experience sampling (ESM) allowed us to gain detailed insights into developers' *individual* productivity perceptions, tasks worked on and interruptions at scale, the method also poses threats to validity. First, we

rely on the accuracy of participants' self-reports who may not always have been able to recall everything accurately, or filling out the pop-ups regularly might be burdensome after a while. To address these concerns, we minimized the number of experience sampling surveys and allowed participants to postpone them in case they appeared at an in-opportune or intrusive moment, as suggested by Csikszentmihalyi and Larson [2014], the inventors of ESM. Furthermore, our various test-runs and studies helped to optimize the experience sampling surveys and make them easier to respond to, such as limiting the number of required clicks and visualizing cues that are helpful to respond accurately. In the surveys and interviews that we performed after the studies to learn about participants' experience and potential issues with the self-reports, we generally got positive feedback about the method. In one study, participants even reported that the periodic self-reports were valuable as they increased their awareness about productivity and work (Chapter 6.6.3).

Construct Validity. The main threat to construct validity stems from the data that we collected automatically using *PersonalAnalytics*, since our monitoring component is limited to data trackers that we build *prior* to the studies. We tried to mitigate this risk by capturing a broad range of data on developers' work, including logging application usage, user input, emails, meetings, web browsing and more. In most cases, we added experience sampling and further collected qualitative data from subjects from surveys or interviews to validate our approach and observations, triangulate the data, and get even deeper insights into their work. Nonetheless, we cannot exclude that there are other factors that influence developers' work, productivity and interruptions.

Furthermore, our quantitative data is limited to developers' work on their computer only. Time they spent away from the computer, *e.g.*, at a break or in a meeting, is not captured. In Chapter 1.5, we discuss how future work could study how taking advantage of other tracking technologies, such as webcams or biometric sensors worn by the users would allow us to capture an even more holistic picture of developers' work. Running a computer monitoring application in real-world scenarios might capture inaccurate data, due to bugs in the logging application

or different infrastructure set-ups. To mitigate these risks, we extensively tested and constantly improved the monitoring application; no major problems were identified during the studies.

Understanding, categorizing and analyzing the collected data poses another threat to the construct validity of our results, since it is not always straightforward to identify or extract all activities or tasks developers perform from the collected data. For example, an email discussion with co-workers on a code component could be categorized as a specific developer task of solving a bug, or to the more general task of answering emails. To mitigate this risk, we observed participants for extended periods of time, asked them to validate the collected data where possible, and carefully discussed our assumptions within the research team, also based on related work.

Another threat to the construct validity of our results is that the first Open Coding step of the analysis was usually performed by one researcher only (*i.e.*, the author of this PhD thesis). To reduce bias and increase construct validity, we usually performed the subsequent steps of Axial and Selective Coding in a team, discussing responses that could not distinctively be mapped, representative quotes, and the relationships between the categories.

1.4 Challenges & Limitations

There are multiple challenges we faced throughout our research. In particular, there are difficulties to generalize the measurement of productivity, and challenges that are inherent to field studies, such as demonstrating long-term behavior change, recruiting participants for long-term studies, and privacy concerns from collecting sensitive data.

Productivity is Individual. A recurring theme in all our studies was that developer productivity is a highly personal matter, due to the many factors that impact developers' work and productivity differently. It is, thus, challenging to develop a *single* model of productivity that is generalizable to all developers. Some of the productivity themes we identified, such as developers' productivity

personas and how productivity changes over the day (morning-, low-at-lunch, and afternoon-person), are a first step towards identifying groups of developers with similar perceptions of productivity (Chapter 1.5).

Measuring Productivity. One implication of the varying impact that the productivity factors can have on developers is that actually measuring productivity poses two main risks and unintended consequences. First, measuring productivity can negatively influence developers' behaviors, either because it pressures them to perform in a certain way to have a high measured "productivity", or it distorts incentives and developers might start to "game" the system, especially if the measure is inaccurate or unfair [Muller, 2018; Treude et al., 2015]. Second, good management requires qualitative insights into team behavior, instead of a quantitative measure [Sadowski and Zimmermann, 2019]. If managers rely on a productivity measurement only, they could make wrong conclusions and bad decisions, instead of understanding how their developers are working and learning what makes them more productive. One simple example is the previously suggested use of lines of code to quantify productivity: developers would just write more code to appear more "productive", but still solve the same problem only. For these two reasons, our work did *not* aim to quantify productivity. Instead, we focused on understanding productivity from the bottom-up, to support *individual* developers with insights into what happens during productive or unproductive times, and to allow the identification of productive self-improvement opportunities that are tailored to the individual.

Demonstrating Behavior Change. Behavior change is a complex, long-term process, progressing over stages and with a high chance of relapse [Klasnja et al., 2011]. Prochaska and Velicer [1997] found that in the field of health sciences a person has to maintain a behavior change for several years to truly stick. Hence, to demonstrate that a system successfully fosters long-term behavior change requires large-scale studies, ideally by performing a *Randomized Control Trial* (RCT) [Chalmers et al., 1981] over multiple years, which is often not feasible with early-stage systems. It is also almost impossible to control for

all the factors that could influence the outcomes of such a field study, and many studies had difficulties to exactly show *why* the behavior changed [Klasnja et al., 2011]. As a result, more recent studies that evaluated the impact of early stage personal informatics (PI) systems, ran *Efficacy Trials* and studied specific outcome measures to test whether the technology accomplishes what it intends to do. In our work, we were careful to not over-claim the effect of our approaches, *e.g.*, *PersonalAnalytics* and *FlowLight*, by stating explicitly that we primarily studied the effect on participants' awareness, and that we observed short-term behavior changes only, which might have been reversed after ending the study.

Recruiting Study Participants. Another challenge of our field studies was to recruit study participants who were willing, able and allowed to install *PersonalAnalytics* on their computers and participate in studies over several days or weeks. First, we were limited by companies that allowed us to study their employees and collect data about their work, and their IT infrastructure, which sometimes made it impossible to install third-party applications. In many cases, approaching the management and allowing the review and analysis of *PersonalAnalytics* prior to the study and applying code signing often reduced concerns. *PersonalAnalytics* was initially only available on the Windows operating system, which further limited the number of candidates for participation. Additionally, monitoring parts of one's work limits our pool of participants to people who are generally open to and interested to learn more about and improve themselves, resulting in a certain self-selection bias that we discuss in Chapter 1.3. Besides extensively testing each version of *PersonalAnalytics*, we therefore spent a substantial amount of time with the recruitment of participants. We approached them via personal contacts, at developer events, and via social media initially, and later cultivated connections with companies to run follow-up studies. We further observed that providing participants with easy access to the collected data and presenting visualizations that can provide insights into their work, were a strong motivator for participation. In many cases, participants showed these insights to their co-workers, who in turn approached us to participate in the study as well.

Collecting Sensitive Data. Monitoring developers' computer interaction at their real-world work poses the risk of collecting sensitive data, which may cause privacy concerns. While more fine-grained data, such as the application content or the text developers type on their keyboards would provide valuable information to even better model developers' work and create more accurate insights, it also might reveal details about the company's products or developers' work and personal life that they are not comfortable sharing. Reducing privacy concerns, thus, required a careful reconciliation of capturing only a minimum set of data that is required by the study, but also tracking as much as possible to get a broad view and understanding of development work. We also made sure to be very transparent and specific about what data is captured, and how it is collected, processed, and analyzed. During the study, participants were always in full control of their data, since they could investigate and modify the locally stored data, disable data trackers they felt uncomfortable with, and review the data before sharing it with us. Surprisingly, after gaining participants' trust, they were often willing to collect even more data, and regularly suggested additional data sources we could implement trackers for.

1.5 Opportunities for Future Work

Our research opens several opportunities for future research, ranging from even further extending our understanding of developers' work and productivity, all the way to addressing productivity of development teams. Chapters 2 to 8 also discuss further opportunities for future work that are more specific to the respective research projects.

Interplay between Individual and Team Productivity. In our work, we focused on modeling and fostering work and productivity of individual developers. One drawback of giving developers insights into only their own productivity is that their behavior changes might have a negative impact on the overall team productivity. As an example, a developer who blocks out interruptions at inopportune times to focus better could be blocking a co-worker who needs to ask a question or clarify things. Hence, and since collaboration accounts for about

one fourth of developers' workdays (Section 2.5.1), a better understanding of the interplay between individual and team productivity is important. We could imagine integrating and aggregating the individual models to come up with team models that provide insights into how the team coordinates and communicates at work, and that foster productive work on a team level. For example, being aware of co-workers' most and least productive times in a workday could help to schedule meetings during times where everybody is the least productive and where interrupting one's work for a meeting has the least negative effect.

Being more aware of the tasks each member of the team is currently working on and how much progress they are making could also be useful for managers or team leads to identify problems early, *e.g.*, a developer who is blocked on a task or uses communication tools inefficiently, and take appropriate action. However, providing teams and managers with relevant and accurate measures is challenging, as this could raise privacy concerns from sharing and aggregating the data, and as a large and ever-growing amount of information and artifacts has to be processed. One way to identify relevant measures of team collaboration is the Model of Regulation by Arciniegas-Mendez et al. [2017], which helps to compare and analyze collaboration practices and tools in software engineering. Furthermore, previous work on team dashboards (*e.g.*, [Brath and Peters, 2004; Jakobsen et al., 2009; Treude et al., 2015]) suggests that aggregating and summarizing the data helps to effectively solve the privacy and information overflow challenges.

More Holistic Picture on Developers' Work and Productivity. Our studies revealed several opportunities to further extend our understanding and gain an even more holistic picture of developers' work and productivity. At the same time, our studies mostly focused on understanding developers' work at their computer. Yet, even the time developers spend *away* from the computer might have a substantial impact on their productivity, especially since we found that developers spend a significant part of their workday not actively using their computer (up to 49%, Chapter 2.5.1), for example during discussions with colleagues or when interacting with other devices such as their smartphone. Therefore, there is potential to further examine and better support work away

from the computer. One way to capture more about the time developers spend away from the computer is through webcams (*e.g.*, [McDuff et al., 2012; Samara et al., 2017]), mobile activity trackers (*e.g.*, [Consolvo et al., 2008b]), or biometric sensors. In one of our own studies [Züger et al., 2018], we used biometric sensors, such as the Fitbit, to capture data that is not directly linked to computer usage.

In multiple studies, participants also stated interest in better understanding how non-work time and private aspects, such as their sleep, exercise or nutrition, impact their productivity. Wearable or biometric sensors could make it possible to capture this, and we might be able to answer devise better support that improves the worklife and productivity of developers.

In addition, we could study the distinctness of the factors that influence developers' work and productivity. Since the identified factors impact developers differently, future work could investigate how pronounced they are, by considering additional variables, such as the company culture and development processes, or developers' gender (*e.g.*, GenderMag [Burnett et al., 2016]) and personality (by using standardized questionnaires from psychology such as BFI-2 [Soto and John, 2016] and UPPS [Whiteside and Lynam, 2001]). Murphy-Hill et al. [2019] recently made a first step towards better understanding the distinctness of productivity factors, by correlating a list of 48 productivity factors with developers' self-rated productivity at three different software companies. They found that the top ten productivity factors are non-technical, and that technical factors, including code reuse and the accuracy of incoming information, had the highest variance between the three companies.

Digital Developer Assistant. A re-occurring feedback by participants in several field studies was that they were interested in receiving personalized recommendations with opportunities for self-improvement. In the future, we plan to study if we can leverage the discovered work habit goals and strategies (Chapter 7), and compare them with developers' *current* work habits, to identify personalized recommendations for self-improvements. For example, depending on whether a developer is a morning or evening person (see [Taillard et al., 1999]), a personal recommendation could suggest to work on the most difficult tasks during times

of the day when the developer is usually most productive and can focus best. Another example is to leverage our task detection model (Chapter 5) to identify and build up task contexts, consisting of work artifacts (*e.g.*, websites or files) and applications the developer used for the task. Upon returning to a suspended or interrupted task, the system could recommend the task context as cues, which previous work has shown to considerably reduce resumption lag and increase productivity [Altmann and Trafton, 2004; Bailey et al., 2001; Rule et al., 2015].

To increase the chance of positive behavior change, these personalized recommendations would also need to be tailored to developers' preferences, for example by altering the timing and content of their interactions with the user. Recent work has suggested that conversational interfaces, such as bots and voice assistants, can successfully adapt to the user's needs by prompting for feedback and integrating into existing work flows [Bradley et al., 2018; Kocielnik et al., 2018; Storey and Zagalsky, 2016; Williams et al., 2018]. We can imagine that such a 'Digital Developer Assistant' could non-intrusively collect data on the developer's current context and work habits, identify personalized recommendations, and then either suggest them through the conversational interface (*e.g.*, recommend to take a work-break when the developer is stuck on a task), or even take certain actions automatically depending on the developer's preferences (*e.g.*, to disable notifications at times of low interruptibility).

Generalization to other Knowledge Workers. Our work focused on understanding and fostering productive work of software developers. We can imagine that many of the findings that we gained and models that we built can be generalized and transferred to the broader range of knowledge workers. With the *FlowLight*, we made a first attempt to explore how our models, in this case on developers' availability for interruptions, generalize to other knowledge workers. The results suggested great potential in targeting the measures and features towards a specific knowledge worker type. While many of the measures and features we included in our models were specific to software development (*e.g.*, debugging time, code reviews) future work could investigate useful features for other knowledge work areas, such as architects, accountants, or other engineers.

1.6 Related Work

Work related to our research can be broadly grouped into research on understanding and quantifying developers' work and productivity, background information on behavior change and goal-setting, and the three persuasive strategies, self-monitoring, self-reflection and external indicators. We provide a comprehensive summary of the related work sections from Chapters 2 to 8 here, and removed them in the respective chapters to reduce repetition.

1.6.1 Software Developer Workdays and Work Habits

Recent work on how developers spend their time at work has focused on specific activities performed in the IDE, the execution of test cases, usage of refactoring features, and time spent on understanding code versus actually editing code [Amann et al., 2016; Beller et al., 2017; LaToza et al., 2006; Minelli et al., 2015]. Other work has investigated developer workdays more holistically, examining how they spend their time on different activities overall, and by employing different methods: observations and interviews [Gonçalves et al., 2011; Meyer et al., 2014; Perry et al., 1994b; Singer et al., 2010; Xia et al., 2017], self-reporting diaries [Perry et al., 1994b], and computer usage tracking [Astromskis et al., 2017]. These studies commonly found that developers spend surprisingly little time working on their main coding tasks, and that the times reported on development and other activities varies greatly. For example, Perry et al. [1994b] found that developers spend about 50% of their time writing code, while Gonçalves et al. [2011] found that it is only about 9%, and the rest is spent with collaboration (45%) and information seeking (32%). Recently, Astromskis et al. [2017] reported the highest fraction of time spent coding (61%) compared to other activities. Reasons for these varying amounts of time spent coding could be differences in how teams organize their work, the processes they follow, and the complexity of the software they develop. Other explanations could be the shift to agile development, which might be why more recent studies generally report higher times spent with collaborative activities, or that the timing of the studies captured an atypical working phase, *e.g.*, when developers were extraordinarily busy before a

deadline or wrapping up a project.

In our work, we further explored developers' workdays by performing several studies, including running a computer interaction monitoring tool at the workplace for several weeks, and using a large-scale survey. Our work confirmed some of these prior findings, such as the little time developers spend on actual coding related activities, and the many collaborative activities they perform. At the same time, it provided a more holistic and more complete picture of how developers' workdays are structured across many companies, and with detailed insights into how they spend their work time on the computer (also outside the IDE), the activities they pursue on typical/atypical and good/bad workdays, and the tasks they work on.

1.6.2 Fragmentation of Development Work

One aspect of developers' work that has drawn much attention is the high fragmentation of their work, which, for instance, is caused by planned meetings, unexpected requests from co-workers, unplanned or blocking tasks, or even just background noise in the office. A large body of research has investigated interruptions—a major reason for fragmentation—in great detail, *e.g.*, the length and types of interruptions, the frequency of self-interruptions versus externally initiated ones, resumption strategies, and their impact on work performance [Cutrell et al., 2000; Czerwinski et al., 2004; Iqbal and Horvitz, 2007; Parnin and DeLine, 2010a; Perlow, 1999; Van Solingen et al., 1998]. For example, Mark et al. [2005] found that 57% of all tasks are interrupted and, thus, are often fragmented into small work sessions, and Chong and Siino [2006] found that most interruptions, including self-initiated ones, lasted around 2 to 3 minutes. While many of these interruptions can be beneficial, for instance to resolve problems quickly [Isaacs et al., 1997], they are especially costly when they happen at an in-opportune moment, such as when the developer is very focused or engaged in a difficult task. Interruptions can further lead to a higher error rate, slow task resumption, higher anxiety, and an overall lower work performance [Bailey et al., 2001; Czerwinski et al., 2000, 2004; Mark et al., 2008a; Parnin and Rugaber, 2011]. Resuming an interrupted task can be challenging, as Parnin and Rugaber [2011] found that

only one out of ten interrupted programming tasks is being continued within a minute after the interruption, and resuming an interruption takes developers on average about 15 minutes. Emails and meetings are two further major sources of distraction and frustration that fragment developers' work. Emails were shown to extend workdays [Mazmanian, 2013] and add stress, especially when developers receive high amounts of emails [Dabbish and Kraut, 2006] or spend a lot of time on them [Barley et al., 2011]. Meetings were also related to a decrease in productivity, especially when they have no clear objective, are not well prepared, or attendees are getting side-tracked [Meyer et al., 2014; Niemantsverdriet and Erickson, 2017].

Supporting previous research, we found that one main reason for work fragmentation is that developers switch frequently between activities and tasks. On average, they spend less than two minutes in an activity (except for meetings), and spend about 13 minutes on a task, before switching to another. Our studies further allowed us to uncover nuances and sort out misconceptions about how various factors impact developers' fragmentation of work and productivity differently. For example, when we looked more closely into when developers are most vocal about meetings and interruptions being one of the main detriments to productive work, we found that their impact on productivity depends on the project phase: during non-development phases, they are better accepted and more productive. Finally, we presented an approach, *FlowLight*, to reduce costly external interruptions at the workplace, and thereby reducing work fragmentation and increasing productivity at work.

1.6.3 Software Developer Productivity

Starting in the 1970s, researchers and practitioners explored multiple ways to quantify developer productivity. Most of these productivity measures calculate the rate of output per unit of input, with the output being based on a single artifact or deliverable and input being time-based. Examples are the number of lines of source code (SLOC) written per day [Devanbu et al., 1996], the number of function points per month [Albrecht, 1979], or the tasks completed per month [Zhou and Mockus, 2010]. Most of these measures only capture a

small part of a developer's work, also impeding the provision of a more holistic picture of developers' work and productivity [Treude et al., 2015]. A more complete list of approaches to quantify productivity using technical factors can be found in our previous work [Meyer et al., 2014]. Researchers have also looked more broadly into the factors affecting the productivity of software developers. Wagner and Ruhe [2008] categorized these factors into *technical*, such as the programming language [Albrecht, 1979], software tools [Chatzoglou and Macaulay, 1997], software size and complexity [Boehm et al., 2000; Brooks Jr, 1995] and product quality [DeMarco and Lister, 2013]; and *social* factors, such as team and turnover [Blackburn et al., 1996; Boehm et al., 2000; Melo et al., 2011; Murphy-Hill et al., 2019], experience and skills [Boehm et al., 2000; Chatzoglou and Macaulay, 1997], and workplace environment [DeMarco and Lister, 2013].

These studies on productivity are generally separate from studies on developers' workdays, and do not consider the individual differences in development work that might affect productivity as well [Humphrey, 1996; Johnson et al., 2003; Meyer et al., 2014; Vasilescu et al., 2016b]. Thus, we studied developers' work practices and their relationship to developers' perceptions of productivity more holistically, while also examining individual differences and commonalities, and how perceived productivity changes over the day.

1.6.4 Characteristics and Habits of Successful Developers

Generally, most software developers are interested in optimizing their own habits and behaviors to improve their productivity and well-being at work [Li et al., 2015; Sharp et al., 2009]. However, we have not been able to find prior work that looked into the goals developers set to improve their work habits and productivity. Previous work also suggests that it is often the managers who set goals for their developers, even though developers would like to have more involvement with setting their own goals [Couger et al., 1990; Enns et al., 2006; Kalliamvakou et al., 2019]. Goals that managers set include usually either concrete features or development tasks, such as shipping a feature on time with minimal bugs; or growth goals, such as increasing expertise, improving team-work or working more independently.

A related area of research looked into characteristics and work habits of successful developers, some of which developers might consider relevant and important to pursue as goals. Amongst other characteristics, successful developers often share similar attributes, such as striving for productivity and efficiency, being self-aware, asking for and offering help and feedback, constantly learning and self-improving, doing data-driven decisions, and setting challenging goals [Baltes and Diehl, 2018; Couger et al., 1990; Graziotin et al., 2015b; Li et al., 2015; Sharp et al., 2009]. Successful developers also manage to find a good balance between focused work and helping others [Baltes and Diehl, 2018; Li et al., 2015].

In our work, we aim to better understand what productive work habits are, as well as what goals and strategies developers employ, to motivate productive behavior changes that increase productivity and well-being.

1.6.5 Fostering Behavior Change with Goal-Setting

Behavior change is a complex and long-term process [Zimmerman, 2006] that was modeled and formalized in multiple theories, such as the Transtheoretical Model of behavior change (TTM) by Prochaska and Velicer [1997], and more specifically to personal informatics, the Stage-Based Model of Behavior Change by Li et al. [2010] and the Lived Informatics Model by Epstein et al. [2015]. TTM models behavior change as a sequence of stages which the person advances through until a behavior change happens and can be maintained. *Awareness increase*, also referred to as consciousness, is one of the processes that lets people advance between stages. In particular, it helps people to advance from being unaware of the problem behavior (TTM's *precontemplation* stage) to acknowledging that the behavior is a problem, and the intention to improve it (*contemplation* stage). There are two ways to identify a problem behavior, either by increasing people's self-awareness or external awareness. Internal, or self-awareness can be increased through self-monitoring or self-reflection, and was shown to be valuable to identify opportunities for positive behavior change, and usually allows people to also measure their progress towards their goals [Consolvo et al., 2009; Kersten-van Dijk et al., 2017]. External awareness can be increased through social incentives, competitions or an external indicator, and was shown to motivate people to not

only reach, but also maintain their goals over prolonged periods of time [Edelson et al., 2011; Fogg, 2003; Fritz et al., 2014; Lin et al., 2006; Rooksby et al., 2014]. The three concepts to increase awareness that we applied in our work, self-monitoring, self-reflection and external indicators, are discussed in detail in the following three sections.

Once people are ready to take action and change their behavior (TTM's *preparation* stage), they define the target behavior or outcome as a *goal* that they work towards (*action* stage) and maintain once it is reached (*maintenance* stage). Latham and Locke, who pioneered *goal-setting* research, identified key principles that improve chances of successful goal-achievement: defining clear and challenging goals, committing and actively working towards them, and measuring progress or getting feedback on goal achievement [Locke and Latham, 1990, 2002] These principles are closely related to Doran's SMART goals approach, describing that goals must be specific, measurable, achievable, relevant and time-bound [Doran, 1981].

1.6.6 Self-Monitoring in the Workplace

An increasing amount of people are using applications and wearable devices to track certain aspects of their lives, in particular related to physical activity [Consolvo et al., 2008b; Lin et al., 2006], emotional states [McDuff et al., 2012], stress [Mark et al., 2016a,b], and sleep [Kay et al., 2012]. This self-monitoring leads to increased self-awareness, which helps to realize bad habits in behavior [Bentley et al., 2013; Choe et al., 2014; Lin et al., 2006] and helps to set personal goals that often promote deliberate or unconscious behavior change (*e.g.*, [Calvo and Peters, 2014; Fogg, 2003; Hollis et al., 2015; Munson and Consolvo, 2012]). For example, physical activity trackers, such as the Fitbit, motivate users to pursue a more active and healthier life-style [Fritz et al., 2014; Lin et al., 2006]. In addition to work on quantifying many aspects of a person's life, there is a growing body of research that focuses on quantifying work and promoting more productive work habits with *automated* self-monitoring techniques. Many of these approaches focus on the time spent in computer applications [Kim et al., 2016; Manictime, 2019; RescueTime, 2019; Slife, 2019; Whittaker et al.,

2016], work rhythms [Begole et al., 2002], or tasks worked on [Dragunov et al., 2005]. Some approaches specifically target the activities of software developers in IDEs, *e.g.*, Codealike [2019], Wakatime [2019] and WatchDog [Beller et al., 2017]. Few of these tools have been evaluated (*e.g.*, [Huang et al., 2016; Kim et al., 2016; Rooksby et al., 2016; Whittaker et al., 2016]), limiting our knowledge of the overall user value of these tools and if they can affect developers' behavior. Previous research also discovered that users rarely engage with the captured data, resulting in a lower awareness and reducing chances of a positive behavior change when using a self-monitoring tool [Collins et al., 2014; Huang et al., 2016; Kim et al., 2016].

In our work, we investigated how we can map the success of self-monitoring systems from the physical activity and health domain to the workplace, and learn more about developers' expectations of and experience with such tools. Specifically, our objectives were to learn how developers use a workplace self-monitoring tool, *PersonalAnalytics*, in practice, and study its impact on developers' self-awareness and behavior.

1.6.7 Self-Reflection in the Workplace

In contrast to self-monitoring, where users *continuously* track certain behaviors, self-reflection describes the process of *purposefully* reviewing and thinking about one's behaviors. Prior research on self-reflection has mostly focused on understanding how particular approaches, such as just-in-time interventions or mobile apps, support goal-monitoring and -achievement, by providing participants with pre-determined goals that they could further personalize and adapt (*e.g.*, defining a specific exercise plan). This research mostly focuses on non-work related areas of life, such as health [Gasser et al., 2006; Monkaresi et al., 2013], food intake [Cordeiro et al., 2015], students' learning behavior [Johnson and White, 1971; Morisano et al., 2010; Travers et al., 2015], and physical activity [Hermanny et al., 2016; Klasnja et al., 2009; Munson and Consolvo, 2012]. However, there are cases, including sleep and work, where pre-determined goals are not suitable for everyone, or they would have varying impacts on different people. In these cases, researchers have looked more specifically into how *purposeful*

self-reflection aids individuals to identify and refine *personal* growth goals that are relevant and important to them. One example is Lee *et al.*'s work on how to design a self-experimentation approach that allows individuals to identify behavior change goals for improving sleep quality [Lee et al., 2014, 2015, 2017]. The researchers provided participants with an open-ended journal to self-reflect on their sleep, a structured approach to define self-improvement goals, and just-in-time interventions to remind them about their goals. They found that the structured self-reflection allows users to self-experiment with goals and identify better behavior change strategies.

There is also research that investigated the value of workplace self-reflection on task completion [Amabile and Kramer, 2011; Claessens et al., 2010], time management [Pammer et al., 2015; Rooksby et al., 2016; Whittaker et al., 2016], and detachment from work [Kocielnik et al., 2018; Williams et al., 2018]. The research has shown that successful self-reflection approaches are often structured, since they guide users' reflection with a predefined set of questions [Kersten-van Dijk et al., 2017; Schwarz and Oyserman, 2001].

In software development, Humphrey has taken a first step towards workplace self-reflection for developers with the Personal Software Process (PSP). PSP focuses on a set of basic metrics that developers reflect on (*e.g.*, time, size, quality, schedule data), to better understand and improve their performance, quality, time estimations and skills [Humphrey, 1996]. Baltes *et al.* recently found that many developers already self-monitor their work, by using tools such as RescueTime [2019], Codealike [2019] or Wakatime [2019], but only very few actively and regularly reflect on work (3 out of 204 survey participants) [Baltes and Diehl, 2018].

Therefore, we were interested in learning how open developers are towards actively self-reflecting on their work on a daily basis during several weeks. We were further interested in learning how self-reflection and goal-setting can be combined to identify personal work habit goals and strategies that foster productive behavior change for developers.

1.6.8 External Indicators in the Workplace

Since interruptions have been shown to be one of the biggest impediments to developers' productive work, we investigated how we can quantify interruptibility using computer interaction data, and how to display interruptibility using an external indicator. Several approaches have been proposed to increase awareness about interruptions and reduce their costs, ranging from simple manual strategies to more advanced automated systems.

One strategy is to postpone interruptions at moments when the interruptee is in the middle of a task to naturally occurring breakpoints—aka. *defer-to-breakpoint strategy*. This idea is based on studies suggesting that the cognitive load drops at task boundaries, and that interruptions at lower cognitive load are less harmful [Bailey and Iqbal, 2008; Borst et al., 2015]. While several approaches that implement this strategy (*e.g.*, Ho and Intille [2005]; Iqbal and Bailey [2008]) are successful at mitigating interruptions from the computer and mobile devices, they do not address the frequent and costly in-person interruptions in workplaces.

A second strategy that builds upon the idea of deferring interruptions to more opportune moments is to *continuously* indicate a person's interruptibility to potential interrupters, and thereby implicitly help negotiating the timing of the interruption. While knowledge workers were shown to already employ simple manual indicators, such as headphones or closing the office door [Sykes, 2011], researchers have also developed approaches that indicate interruptibility based on automated measurements. Most prior work focused on contact-list-style tools, such as ConExus [Tang et al., 2001] and Lilsys [Begole et al., 2004] that are installed on the user's computer and vary mostly in which data they use to quantify interruptibility. MyTeam, an approach by Lai et al. [2003], uses data on presence, network connection and user input to indicate availability in a contact list. Overall, study results for these computer-based interruptibility indicators suggest that they increase the awareness about the disruptiveness of interruptions, but do not reduce in-person interruptions.

Only few researchers examined external indicators so far, the most similar to the *FlowLight* being work by Bjelica et al. [2011]. In their work, they developed an automated indicator visualizing interruptibility through ambient lighting

effects based on detected activities (*e.g.*, low interruptibility during meetings). In our work, we developed an *automated* interruptibility measure that is based on computer interaction data and combined it with a physical indicator in the form of a traffic-light like LED that we placed on participants' desks. Thereby, our approach is more direct and prominent than subtle ambient lighting. We conducted a large-scale and long-term user study to investigate the impact and value of *FlowLight* to foster productive work by reducing intrusive in-person interruptions.

1.7 Summary of Contributions

The objective of this research was to foster software developer productivity. A first step towards the objective was to gain a better understanding of developers' work and productivity from a bottom-up perspective (RQ1). The findings from the performed studies informed the development of several models describing developers' work and productivity, which we integrated into tool support. We evaluated these models through various long-term field studies with professional software developers, who used our tools in the field and reported self-improvements in work habits, experiencing fewer costly interruptions, and increased productivity, amongst other benefits. These results support our *Hypothesis* and provide evidence that it is possible to apply models that are based on computer interaction at the workplace, to foster productive work through the provision of self-monitoring (RQ2a), reflective goal-setting (RQ2b), and an external indicator (RQ3).

The contributions of this work are as follows:

- we present empirical findings from a series of studies on software developers' workdays and the factors that influence them, activities and tasks that fragment development work, developers' perceptions of productivity, and the relationship between activities at work and productivity;
- we present *PersonalAnalytics*, a system to capture developers' work and productivity in the field, and to support self-monitoring. Our multi-week

field study showed that *PersonalAnalytics* can foster productive behavior changes, and revealed design recommendations for self-monitoring systems;

- we present the results of a field-study in which our reflective goal-setting approach allowed developers to identify individual work habit goals, and motivated productive self-improvements at work;

- we present and evaluate a model based on computer interaction data to automatically detect developers' task switches and types, and show that we can achieve high accuracy and only a short delay;

- and, we present the *FlowLight*, an approach to sense and externally indicate interruptibility based on computer interaction. Our large-scale field study shows that the *FlowLight* successfully increases external awareness about interruption cost and reduces interruptions at in-opportune moments.

  Finally, we open-sourced *PersonalAnalytics* for reproducibility and future research, and licensed the *FlowLight* to be released as a commercial product.

1.8 Thesis Roadmap

The remainder of this thesis consists of 7 chapters, each based on a publication published at an internationally renowned, peer-reviewed conference or journal. An overview of all publications, including publications that were outside of the scope of the thesis, is given in Figure 1.5.

Chapter 2 addresses RQ1a, RQ1c and RQ1d, and investigates the activities developers pursue during their workdays, how they fragment their work, and their relationship with perceived productivity. My contributions to this chapter are the design and execution of the field study, participant recruitment, tool development, partial data analysis, and paper writing.

Chapter 3 addresses RQ1d, explores differences and commonalities in developers' productivity perceptions, and describes groups of developers with similar perceptions as developer personas. My contributions to this chapter comprise the

design and execution of the study, participant recruitment, partial data analysis, and paper writing.

Chapter 4 addresses RQ1a, and extends our understanding of developer workdays, in particular the factors that influence what makes workdays good and typical, workday types, and how collaboration impacts these days. My contributions to this chapter are the analysis of the collected data, and paper writing.

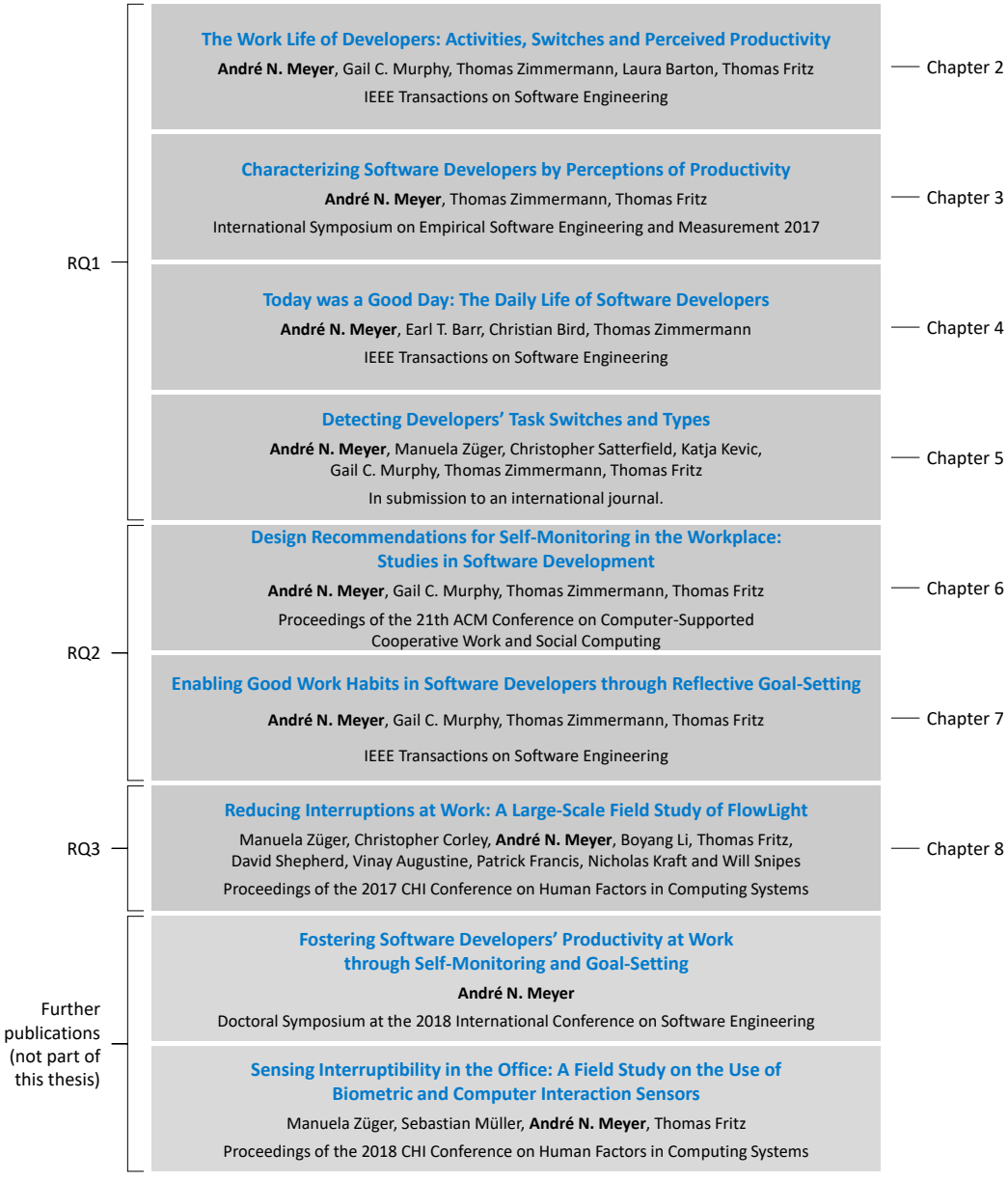
Chapter 5 addresses RQ1b, and allows us to automatically detect developers' task switches and task types based on their computer interaction. My contributions in this chapter are the study design and execution, participant recruitment, tool development, partial data analysis, and paper writing.

Chapter 6 addresses RQ2a and presents a workplace self-monitoring approach, *PersonalAnalytics*, that increases developers' awareness about work and productivity through retrospection and experience sampling. My contributions to this chapter comprise the design and execution of the field study, participant recruitment, self-monitoring tool development, data analysis, and paper writing.

Chapter 7 addresses RQ2b and provides a framework for reflective goal-setting that supports the identification of work habit goals and actionable strategies for productive self-improvements. My contributions to this chapter are the design and execution of the field study, participant recruitment, data analysis, and paper writing.

Chapter 8 addresses RQ3, introduces an approach to automatically measure the interruptibility based on computer interaction, and presents the *FlowLight* to increase external awareness about co-workers' interruptibility using an external indicator. The approach reduces costly interruptions at inopportune moments, and thus, improves productivity. My contributions to this chapter comprise involvement in all main parts of this large team project, including the tool development, execution of the field study, data analysis, and paper writing.

Figure 1.5: Thesis roadmap.



2

The Work Life of Developers: Activities, Switches and Perceived Productivity

André N. Meyer, *Laura E. Barton, Gail C. Murphy, Thomas Zimmermann, Thomas Fritz*

Published in the 2017 IEEE Transactions on Software Engineering Journal

Contribution: Study design and execution, participant recruitment, tool development, data collection, partial data analysis, and paper writing

Abstract

Many software development organizations strive to enhance the productivity of their developers. All too often, efforts aimed at improving developer productivity are undertaken without knowledge about how developers spend their time at work and how it influences their own perception of productivity. To fill in

this gap, we deployed a monitoring application at 20 computers of professional software developers from four companies for an average of 11 full workdays *in situ*. Corroborating earlier findings, we found that developers spend their time on a wide variety of activities and switch regularly between them, resulting in highly fragmented work. Our findings extend beyond existing research in that we correlate developers' work habits with perceived productivity and also show productivity is a personal matter. Although productivity is personal, developers can be roughly grouped into morning, low-at-lunch and afternoon people. A stepwise linear regression per participant revealed that more user input is most often associated with a positive, and emails, planned meetings and work unrelated websites with a negative perception of productivity. We discuss opportunities of our findings, the potential to predict high and low productivity and suggest design approaches to create better tool support for planning developers' workdays and improving their personal productivity.

2.1 Introduction

A software developer's work day might be influenced by a wide variety of factors such as the tasks being performed, meetings, interruptions from co-workers, the infrastructure or the office environment (*e.g.*, [DeMarco and Lister, 1985; Perry et al., 1994b; Singer et al., 2010]). Some of these factors result in activity and context switches that can cause fragmented work and that can have a negative impact on the developer's perceived productivity, progress on tasks, and quality of output (*e.g.*, [Boehm, 1987; Meyer et al., 2014]). As a result, researchers and practitioners have both had a long interest in better understanding how developers work and how their work could be quantified to optimize productivity and efficiency.

Researchers have investigated work practices and work fragmentation in detail from various perspectives, specifically the effect of interruptions on fragmentation (*e.g.*, [Czerwinski et al., 2004; Iqbal and Horvitz, 2007; Parnin and Rugaber, 2011; Van Solingen et al., 1998]) and how developers organize their work in terms of tasks and working spheres (*e.g.*, [González and Mark, 2004; Meyer et al.,

2014]). Using both a diary and an observational study format to understand software developer work practices, Perry et al. [1994b] gained several insights, including that most time was spent coding, and that there was a substantial amount of unplanned interaction with colleagues. Singer et al. [2010], using several study methods including tool usage statistics, found that developers spent most of their time reading documentation and that search tools were the most heavily used. Since the time these earlier studies on developers' work practices were conducted, empirical studies of software development have focused more on particular aspects of a developer's work day. For example, Ko et al. [2007] observed software developers to determine what information was needed to perform their work and how they found that information. Other studies have focused on how developers spend their time inside the Integrated Development Environment (IDE) (*e.g.*, [Amann et al., 2016; Minelli et al., 2015]). The industry has also seen an increasing trend with self-monitoring tools to track activity and work habits, with applications such as RescueTime [2019] or Codealike [2019].

Starting in the 1970s, researchers have also been exploring various different ways to quantify a developer's productivity. Most of these identified productivity measures capture a small part or single aspect of a developer's work, such as the number of tasks per month [Zhou and Mockus, 2010], the number of lines of code written [Blackburn et al., 1996], or the resolution time for a modification request [Cataldo et al., 2008]. However, these studies on productivity are generally separate from studies of work fragmentation and of how developers work. Furthermore, these measures do not take into account the individual differences in development work that might affect productivity as pointed out by previous work [Humphrey, 1996; Johnson et al., 2003; Meyer et al., 2014; Vasilescu et al., 2016b].

In this paper, we study developers' work practices and the relationship to the developers' perceptions of productivity more holistically, while also examining individual differences. In particular, our study seeks to answer the following research questions:

RQ1: What does a developer's work day look like?

RQ2: How fragmented is a developer's work?

RQ3: Are there observable trends in how developers perceive their productivity?

RQ4: What is the relationship between developers' activity and perceived productivity at work?

To investigate these questions, we designed and conducted a study involving the monitoring of 20 developers' interactions with their computer over a two week time period. From this monitoring, we were able to gather logs describing how a developer was interacting with the computer (i.e., through the keyboard or the mouse) and in what applications the interaction was occurring. Our monitoring also gathered self-reports from the developers about their current task(s) at 60 minutes time intervals, and a self-rating of their perceived productivity. The 20 developers from whom we gathered data worked for 4 different companies of varying size, with varying projects, project stages and customers, providing more diversity in our results than has been available in previous holistic studies. This approach also allows us to see whether earlier research findings, such as how much time developers actually spend coding [Perry et al., 1994b] and typical coding related activities [LaToza et al., 2006], hold in contemporary development, and to enhance emerging theories about fragmented knowledge work [González and Mark, 2004].

Based on our analysis of the gathered data, we observed that productivity is a highly personal matter and perceptions of what is considered to be productive are different across developers. No one model we built relating actions and activity to perceived productivity was able to explain a large number of developers. However, we did find that many developers consider email, planned meetings and work unrelated browsing as less productive activities, and usually perceive themselves as more productive when they have a higher user input rate as measured by mouse clicks and keystrokes. Further, developers' work is highly fragmented, as developers are spending only very short amounts of time (0.3 to 2 minutes) in

one activity before switching to another one. Even though we observed that some aspects of a developer's work habits are highly individual we found consistent trends across multiple people. For example, some developers parcel their work out over a longer time span, while others choose to contain their work time and stay off of the computer during the evening. Some seem to be morning people, with higher productivity ratings in the morning hours, others are afternoon people. Finally, we discuss implications and opportunities of our findings to help improve and predict developer productivity.

This paper provides the following contributions:

- it provides insights into software developers' work habits, including the frequency and duration of performing particularly activities and application use;
- it provides data about the rhythms in developers' perceived productivity, which opens opportunities for retrospective tools and recommender systems for when developers might best perform particular activities;
- it demonstrates that productivity patterns for individuals are consistent, but vary when comparing across groups of software developers; and,
- it shows that perceived productivity and the factors that influence it, such as emails, meetings, or activity switches, are highly individual.

Section 2.2 presents relevant related work on developers' work practices, the high fragmentation of their work, approaches on quantifying development activities and on measuring productivity. Sections 2.3 and 2.4 describe the study method employed and the data collected. Section 2.5 presents the results of our study in terms of what a developer does, the fragmentation of a developers' work, the rhythms of a developer's perceived productivity and which activities and actions a developer perceives as productive. Section 2.6 outlines the threats to our results. Section 2.7 discusses implications and opportunities of the findings of our study for future tool support and presents results of a preliminary analysis on predicting two levels of productivity. Section 2.8 summarizes the paper.

2.2 Related Work

Related work can broadly be classified into four categories: developers' work practices, work fragmentation, the quantification of development activities and productivity. To avoid repetitions in this thesis, we present and summarize the related work in Section 1.6 of the synopsis.

2.3 Study Method

To answer our research questions, we conducted an *in situ* study at four international software development companies of varying size. We collected data from 24 professional software developers using a combination of experience sampling (diary study) and a background monitoring application. The monitoring application logged a wide range of digital activities over several weeks with detailed precision. Experience sampling was used to collect participants' perceptions of productivity, as well as self-reported tasks and activities they performed throughout their work day.

2.3.1 Participants

We used personal contacts, emails and sometimes a short presentation at the company to recruit participants. Of the total 24 participants, we discarded the data of 4 participants as they did not respond to a sufficient amount of experience samples. Two participants responded to less than 5 experience samples over the course of the study, as they thought it was too intrusive for their work. The other two participants responded to very few samples since they were either working on a different machine as the one initially indicated or did not use their machine for more than an hour per work day.

Of the remaining 20 participants, 1 was female and 19 were male. All participants are professional software developers, with varying roles between individual contributors[1] and lead. At the time of the study, our participants

[1]We defined an individual contributor as an individual who does not manage other employees.

had an average of 14.2 years (±9.6, ranging from 0.5 to 40 years) of professional software development experience and an average of 18.9 years (± 9.2, ranging from 1.5 to 40 years) of total software development experience, including education. An overview of our participants can be found in Table 2.1.

To capture various kinds of software development practices, we sampled developers from 4 different companies of varying size, in different locations and project stages, using different kinds of programming languages, and with different kinds of products and customers. Companies resided either in the USA (company A), Canada (company B and C) or Switzerland (company D). The company sizes varied from less than 10 developers (company D), to a few hundred (company C), and thousands of developers (company A and B). The project stages varied from working on initial releases (company D), over working on a next big release (company D and B) to being in a project maintenance cycle (company A and C). The developers in company A were mainly programming in C++ and C#, in company B in Java and C#, in company C in JavaScript, C# and SQL, and in company D in JavaScript, Java, C# and SQL. The products developed by the companies range from developer support tools, to power monitoring and robotics software, all the way to cloud-solutions for B2B and B2C customers.

2.3.2 Procedure and Monitoring Application

The monitoring application was developed and tested to run on the Windows 7, 8 and 10 operating system. To make sure it works properly, the collected data is accurate and to optimize the performance, we deployed the tool in multiple steps. After an initial phase of extended testing on 3 researchers' machines, we deployed it to three developers in company B and one developer in company C over several weeks, to ensure correct functionality in many different computer set-ups and use cases and to ensure the tool is stable and reliable enough for daily use.

We then installed the monitoring application on the first day of the study after a presentation that included an introduction to the study and details on the data that was being collected with the monitoring application. Participants were assured of the anonymity and privacy of their data and were shown the location

Table 2.1: Study Participants (IC: Individual Contributor, Distribution is on a 7-point Likert scale: left = "not at all productive" (1), right = "very much productive" (7)).

Participant			Total Dev.	Prof. Dev.	# Work	Perc. Prod. Ratings	
ID	Comp.	Role	Experien.	Experien.	Days	#	Distribution
S1	C	IC	15	10	8	45	
S2	C	IC	25	18	8	101	
S3	C	IC/Lead	23	16	9	62	
S4	C	IC	20	15	8	94	
S5	C	IC	19	15	8	40	
S6	C	IC/Lead	20	20	8	62	
S7	C	IC	16	11	15	80	
S8	C	IC	40	40	11	73	
S9	C	IC	29	29	12	89	
S10	C	IC	22.5	19	12	92	
S11	B	Lead	14	6	9	51	
S12	B	IC	1.5	0.5	7	40	
S13	D	Lead	23	12	10	76	
S14	D	IC	8	4	9	88	
S15	D	IC	5	1	9	71	
S16	A	IC	20	15	11	42	
S17	A	IC	19	5	17	62	
S18	A	Lead	17	17	16	53	
S19	A	IC/Lead	33	23	20	100	
S20	A	IC	8	7	13	30	
Average			**18.9**	**14.2**	**11.0**	**67.6**	

where the logged data was stored on their computer to give them full control over their data. Participants had the opportunity to censor parts of the collected data, which was reportedly done a few times, *e.g.*, when participants were using their private e-banking. The monitoring application logged the currently active process and window title every 10 seconds, or an 'idle' entry in case there was no user input for longer than 10 seconds. In addition an event for each mouse click, movement, scrolling, and keystroke was logged. For keystrokes, we avoided implementing a key logger and only logged the time-stamp of any pressed key.

Perceived Productivity Self-Reports. To capture how developers perceive

Figure 2.1: Notification to Prompt Participants to Respond to the Experience Sampling Survey.

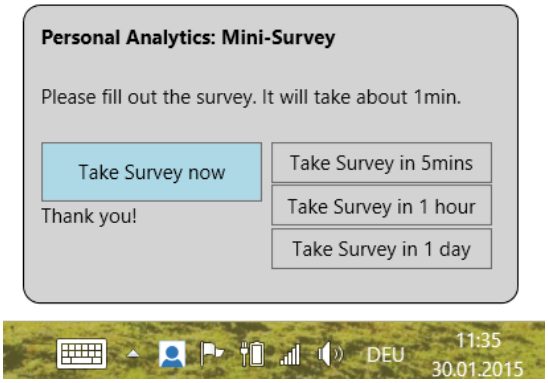
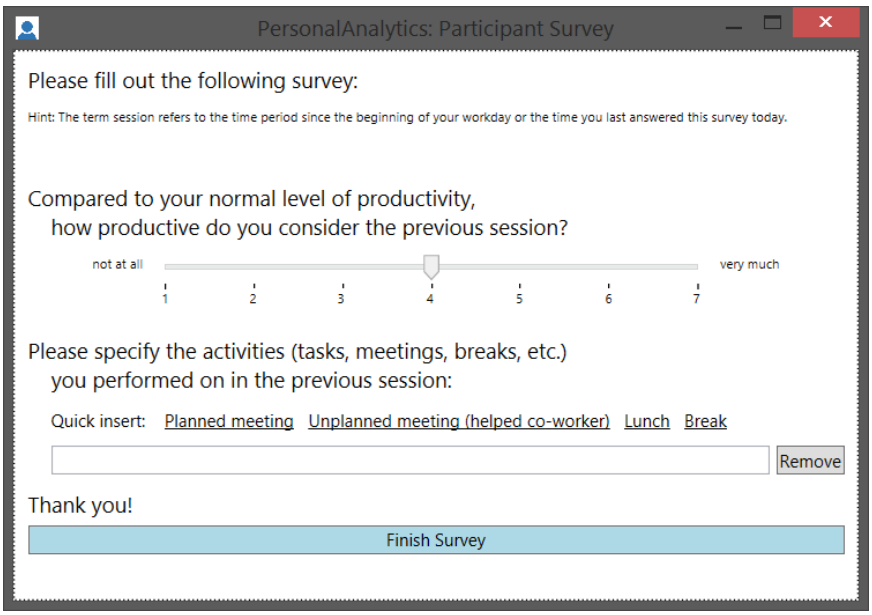


Figure 2.2: Screenshot of the Experience Sampling Survey (Pop-Up).



their own productivity, we used experience sampling in the form of regular self-reports. Experience sampling has previously been used in multiple studies (*e.g.*, [Czerwinski et al., 2004; Mark et al., 2016a,b; Mathur et al., 2015]), and allowed us to capture perceived productivity on a periodic and fine granular basis. For the experience sampling, a notification window appeared on the bottom

right corner of the participants' main screen in regular intervals (see Figure 2.1) prompting the participant to answer a short survey on productivity. To minimize the risk of negatively influencing the participant's work, the notification window also offered options to postpone the survey. By default, the notification window was shown every 60 minutes, but participants could also change the interval, which was only done by one participant who changed it to a 90 minute time interval. Additionally, participants were also able to manually trigger the survey in case they wanted to answer the questions more often. If the participant was regularly working on a virtual machine or secondary device, we installed the application on the other devices as well, but disabled the pop-ups so as to not interrupt the participant more than necessary. Overall, participants answered 74.6% of the triggered self-reports within five minutes of the notification, 21.6% were postponed for an average of 38 (±43) minutes, and 3.8% were ignored.

Once the participant started the self-reporting, another window with the survey appeared (see Figure 2.2). This survey asked participants about their perceived level of productivity for the previous work session using a 7-point Likert scale and to specify the activities and tasks they performed. To facilitate the participant's response, the text boxes offered auto-completion and quick-insert buttons for frequently used and previously inserted activity descriptions. We only used a single question on productivity to minimize the disruption and also since previous research has shown that participants interpret different terms of productivity, such as efficiency, effectiveness, or accomplishment, very similar [Mark et al., 2016b]. Since the survey questions remained the same throughout the study and were related to the current context, we expected the cognitive burden on the participant and the distraction for answering the survey to be relatively low, as also illustrated in other research [Czerwinski et al., 2000; Monk et al., 2008]. Participants used an average of 33.5 (± 39.4) seconds to answer the two questions. We found no significant differences in the total number of times participants answered the survey per day over the whole course of the study, suggesting that they used similar effort throughout the study and the burden of answering did not increase for them.

Table 2.2: Data Collected by the Monitoring Application.

Data	Description	Data Collected
Background Monitoring		
Program usage	current process name and currently active window title, captured once every 10 sec	1 479 383 items
User Input		
- Mouse clicks	as event happens	798 266 clicks
- Mouse movement distance	aggregated pixels moved per sec	2 248 367 entries
- Mouse scroll distance	aggregated pixels scrolled per sec	296 763 entries
- Keystrokes	only count, not exact keystrokes (for privacy reasons)	3 515 793 keystrokes
Survey Pop-Ups		
Tasks and Activities	participants self-reported tasks and activities from the past 60 mins (for one participant: 90 mins)	2 237 items
Perceived Productivity	slider where the participant rates the perceived productivity of the previous work session	1 350 ratings

Procedure. After we explained the study and installed the application, we asked participants to resume their normal working habits during the period of the study and answer the experience sampling probes when the pop-up appeared. We also told participants to ask us any questions about the study, the application or captured data at any point in time during the study. At the end of the study, we interviewed each participant to collect demographic information and information on the project they were working on, the company and their experience of using the monitoring tool and participating in the study. We then collected the monitoring and self-report data and un-installed the application. Table 2.2 summarizes the data we collected from the participants.

2.4 Data Collection and Analysis

During the study, the monitoring application collected data from 2197 hours of participants' computer use over a total of 220 work days. Table 2.1 shows

that each participant was part of the study for between 7 and 20 work days (mean: 11.0 days, ±3.6). Table 2.2 shows how much of each type of information we collected. This section describes how we prepared the collected data for the analysis.

2.4.1 User Input Data

Whenever a study participant pressed a key, or clicked, scrolled or moved their mouse, the event-type and its timestamp were recorded by the monitoring application. We divided the events into work days, where a day spanned from 04:00:00 A.M. to 03:59:59 A.M. the following morning, very similar to Amann et al. [2016]. This division was chosen as some participants stayed up late and logged input continuously before and slightly after midnight, suggesting that their 'work day' was not yet over at that time. With respect to time usage, but not application usage, we removed weekends, as we wanted to examine trends and based on initial inspection the weekend data was highly irregular. If a day had less than 10 minutes of recorded active input for a given developer, that day was not included when considering keyboard and mouse input.

Within a day, we determined *inactive* and *active* periods of computer use. An *inactive* period is a lapse in any type of input for 2 minutes or longer; conversely, an *active* period is 2 minutes or longer during which there is no inactivity. This 2 minute mark was chosen as a reasonable boundary based on previous research that found an average time of 2 min 11 sec being spent on any device or paper before switching [González and Mark, 2004], as well as research by Chong and Siino [2006] who found that a typical interruption for one work team lasted 1 min 55 sec, and 2 min 45 sec for another team. We further defined inactive periods by sorting them into two categories: *short* and *long* breaks away from the computer. A short break from the computer is defined as a period of inactivity that lasts at least 2 minutes but less than 15 minutes and that is often spent with answering a co-workers question or as a coffee break. A long break from the computer is defined as a period of inactivity equal to or longer than the 15 minute threshold, and is often used for meetings and longer breaks from work, such as a lunch [Epstein et al., 2016b; Sanchez et al., 2015]. We used a 15

minutes threshold based on previous work by Sanchez et al. [2015] that described a typical developer's coffee break to be 12 minutes on average and factored in a 3 minute deviation.

2.4.2 Preparing Program Data and Mapping to Activities

Our monitoring application recorded the current process and window titles of participants' program usage during their work day, once every 10 seconds. To provide a higher-level, aggregated view on participants' work, we mapped each entry of a program's use to activity categories, also taking into account the window titles for clarification. These activity categories group actions undertaken by a developer, for example the category *Coding* denotes any developer action that is related to reading, editing or navigating code. We reused the activity categories we identified with an open-coding approach in our previous study [Meyer et al., 2014]. This mapping process was a semi-automated open-coding process; automated, where an action fit into an obvious category (*e.g.*, SourceTree belonging to the activity category *Version Control*) and manual, where actions could not automatically be classified distinctively using the program and window title names (*e.g.*, website names to the activity categories *Work Related Browsing* or *Work Unrelated Browsing*). When a participant did not switch programs or have any mouse or keyboard interaction for the past 10 seconds, the application logged it as 'idle'.

In order to complete the coding, we first defined a set of keywords for each activity that distinctly map a program to an activity. For instance, we mapped Microsoft Word to *Reading or Writing Documents* and Microsoft Outlook either to *Email* or *Planning*, depending on the details in the window title. We created a script using these keywords to produce the initial mapping, then inspected the results and iteratively refined the keywords until most programs were mapped. In cases where a program could be mapped to two or more activities, we performed a manual mapping. For example, as work in a text editor could be mapped to several activities (*e.g.*, *Coding*, *Reading or Writing Documents*, or *Planning*), we manually mapped all editor use. Similarly, we mapped most website use manually, either to *Work Related Browsing* or *Work Unrelated Browsing*. In both

cases, the window title held valuable information about the activity a developer was performing. We further manually checked all automated mappings for the following activities: *Debugger Use*, *Code Reviews*, and *Version Control*. All entries, related to coding, which could not distinctively be mapped to one of these categories, were mapped to *Coding*.

To better understand what participants were doing when they were not actively working on their computer ('idle' events), we combined the data logged by the monitoring tool with participants' self-reports. As most participants not only reported the tasks they worked on in the past work session, but also planned and informal meetings, and lunch and coffee breaks, we could in many cases infer if a period of 'idle' time belongs to one of these categories. In cases where the participant did not report a meeting or break, we had no way of identifying the reason for the 'idle' time, which is why the amount of time spent with planned and informal meetings might be higher than reported in this paper. The self-reports were not only used to map 'idle' time, i.e. time not actively spent on the computer, to breaks and meetings, but also to analyze developers' self-reported tasks.

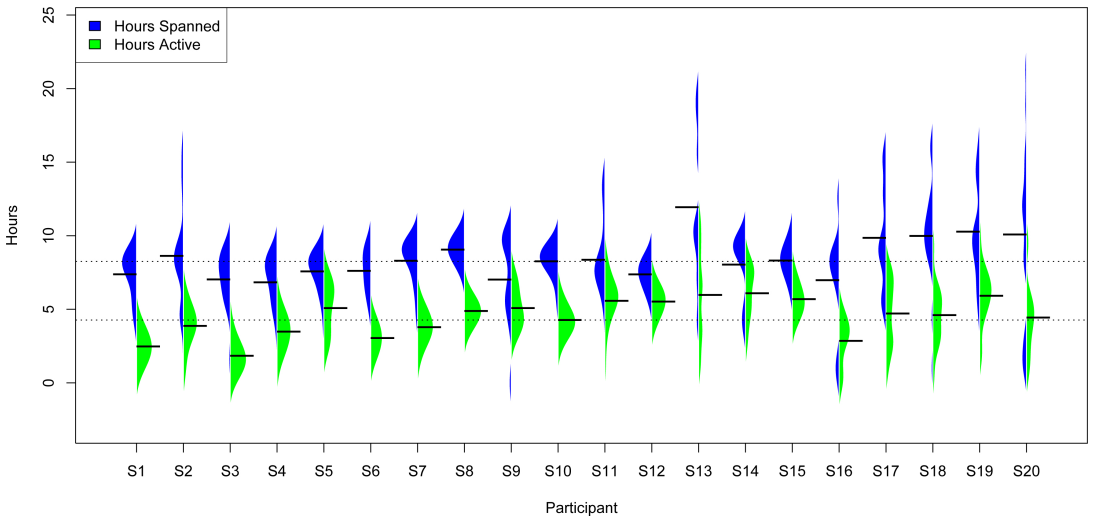
2.5 Results

This section presents the results of our study on developers' work practices and their relation to the perceived productivity by investigating our four research questions.

2.5.1 What Does a Developer Do?

To answer our first research question, "What does a developer's work day look like?", we analyzed the span of a developer's work, the amount of time during the span the developer was active, the nature of the breaks taken, the applications used, and the activities pursued.

Figure 2.3: Total Hours of Work versus Hours Active.



Hours Spanned and Hours Active

The number of *hours spanned* by a developer's work day is defined as the time between the earliest and latest inputs on a given day, regardless of intervening activities or inactive periods. For example, if the first mouse click happened at 9:00 A.M. and the last keystroke happened at 17:30 P.M., the time span would be 8.5 hours. The number of *hours active* is the cumulative time when a participant was logging active input.

Figure 2.3 contrasts the hours spanned and hours active per developer across all days monitored. Some participants (*e.g.*, S13 and S20) tend to space their work out over as many as 21.4 hours, whereas others (*e.g.*, S12 and S15) keep more compact work hours and remain active during the bulk of their time. Overall, developers averaged spans of 8.4 ($\pm$1.2) hours per day, with active time of 4.3 ($\pm$0.5) hours. It should be noted that the hours active are not synonymous with an individual's total working time; since the hours active value is based on the time the participant is using their mouse or keyboard, it does not account for meetings or other work activity away from the computer.

Short and Long Inactive Periods

Every hour, developers take an average of 2.5 (±0.8) short breaks that are about 4.2 (±0.6) minutes long each and in which the developers are not interacting with their computer. This results in a total of 10.5 minutes of inactive time every hour of work, which we assumed to be likely unplanned interruptions, such as co-workers asking a question or a quick coffee break. According to Minelli et al. [2015], who analyzed how developers spend their time inside the IDE, inactive times are often spent with program comprehension. This notion of taking a few minutes to understand, think about or read the current artifact, such as code, a website or document, is likely another reason for these short inactivities. There was no obvious trend towards taking more or fewer short breaks during a particular part of the day; rather, short breaks appear to be fairly evenly distributed for all participants.

The number of long breaks in which developers did not interact with their computer for longer than 15 minutes averaged 3.3 (±1.4) per day, with a total length of 54.7 (±28.2) minutes; this corresponds to an expectation of two longer coffee breaks, a lunch, and perhaps a meeting. The participants with a high number of long breaks appear to be those who have a tendency towards longer hours spanned, with additional long breaks happening in the late afternoon or evening before these individuals returned to work. Since these long breaks were likely planned by the developers, they were also more likely to be self-reported as a distinct activity in the pop-up surveys.

Applications Used and Frequency of Use

Participants used a total of 331 different applications, with each participant using an average of 42.8 (±13.9) different applications over the study duration and 15.8 (±4.1) applications per day.

Table 2.3 shows the ten most popular applications across all participants (all were using Windows operating systems). There is a notable amount of time spent using File Explorer, although this may be due to the fact that it is the only application, other than Internet Explorer, that was used by all 20 participants.

Table 2.3: Top 10 Used Applications (Sorted by Usage).

Application	% of time used	# of users
Microsoft Outlook	14.2%	18
PuTTY	12.8%	8
Google Chrome	11.4%	16
Microsoft Internet Explorer	9.4%	20
Microsoft Visual Studio	8.3%	13
File Explorer	6.6%	20
Mozilla Firefox	5.9%	8
Eclipse	3.0%	10
Microsoft OneNote	2.3%	9
Command Line	2.2%	16

Despite everyone using Microsoft Internet Explorer, 80% of the participants also used Google Chrome and spent more time in it than in the Internet Explorer.

Activities Pursued

Table 2.4 shows the activities developers pursue during their work days. A developer's typical work day is mostly spent on coding (21.0%), emails (14.5%), and work-related web browsing (11.4%). Using the debugger, reviewing other developers' code, and version control account for just 2.4% of the participants' time. When looking at individual versus collaborative activities, 24.4% of a developer's day is spent pursuing collaborative activities with co-workers, customers, or managers, such as planned or ad-hoc meetings and emails. These percentages do not include uncategorized inactive time towards the total time as the monitoring application could only capture non-computer work in case the participants self-reported it.

Table 2.4: Developers' Fragmented Work: Activities Performed.

Activity Category		**% of time over whole period**	**Duration per day (in hrs)**			**Time spent before switching (in mins)**		
			Avg	Stdev	Max	Avg	Stdev	Max
Development								
Coding	reading/editing/navigating code (and other code related activities)	21.0%	1.5	±1.6	7.3	0.6	±2.6	135.7
Debugger Use	using the debugger inside the IDE	0.4%	0.1	±0.2	0.8	0.5	±0.8	13.4
Code Reviews	performing code reviews	1.3%	0.3	±0.4	2.1	1.3	±4.5	13.4
Version Control	reading/accepting/submitting changes	0.7%	0.1	±0.3	2.2	0.6	±1.0	12.9
Email	reading/writing emails	14.5%	1.1	±1.3	8.1	0.9	±4.8	89.6
Planning	editing work items/tasks/todos; creating/changing calendar entries	4.8%	0.5	±1.1	5.1	1.1	±2.5	67.5
Read/write documents	reading/editing documents and other artifacts, e.g. pictures	6.6%	0.5	±0.7	4.5	0.8	±3.3	114.7
Planned meeting	scheduled meeting/call	6.5%	1.0	±1.3	7.1	15.8	±35.3	203.1
Informal meeting	ad-hoc, informal communication; e.g. unscheduled phone call / IM, or colleague asks a question	3.4%	0.5	±0.6	4.2	2.0	±6.5	138.2
Work related browsing	Internet browsing related to code/-work/task	11.4%	0.8	±1.3	12.8	0.5	±5.5	102.6
Work unrelated browsing	Internet browsing work unrelated	5.9%	0.5	±0.7	3.4	1.1	±4.3	91.8
Other	Anything else; aggregates several small sub-categories, such as changing music, updating software, using the file explorer or having a break	11.4%	0.8	±1.4	10.5	0.4	±5.6	112.5
Other RDP	Remotedesktop use which could not be mapped to another category	12.0%	1.5	±1.8	8.2	0.3	±2.6	85.4

When we inspected the data, we observed a notable range between the minimum and maximum time developers spent on each activity per day. The minimum time was virtually 0, or only a few seconds per day. The maximum time a participant spent on a single activity was 12.8 hours on one day: S13, who was evaluating various continuous integration and build systems. It is clear that the duration and type of activities vary greatly depending on the individuals and their current goals. We also observed differences in the distribution of the activities between companies. For example, developers S11 and S12, both from the same company, spent significantly less time on emails, on average just 0.7 minutes ($\pm$0.5) per day, compared to the other participants, who spent an average 74.3 minutes ($\pm$74.8) on emails.

The amount of time spent in coding or debugging might be higher than reported, as it was not possible to map all activities from the category Remote Desktop (*OtherRdp*), as there was not enough context available for an accurate mapping either automatically or manually. Similarly, the amount of time spent in planned and informal meetings might be higher than reported, as participants likely did not self-report all meetings they attended via the pop-up survey. This made it impossible to map all 'idle' time to an activity.

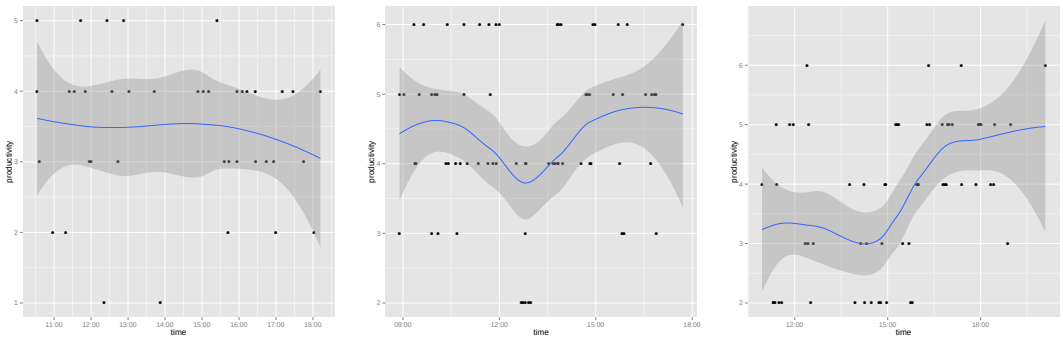
Developers' Self-Reported Tasks

An analysis of developers' self-reported tasks shows that there is a wide variety in what developers work on, but in particular also, in what developers denote as a task. In many cases, participants reported activities they were performing, such as "coding", working on "emails" or performing a "web search", rather than the intention of the activity, such as the change task or the bug they were trying to fix. Only in very few cases, did participants mention a bug ID on which they were working. Furthermore, reported and worked on tasks varied in their granularity. While some participants were working on a task only a single time for a part of the 60 to 90 minutes time window, others reported to work on the same task for several days.

Overall, due to this variance in task definition and granularity, the self-reported tasks did not provide much further insights into developers' work days

Figure 2.4: 3 Types of Developers and their Perceptions of Productivity over the Course of a Work Day.

(a) Morning Person (b) Low at Lunch Person (c) Afternoon Person



other than help with disambiguation of our mappings in some cases. Also, while the number of resolved or worked on tasks has been rated as a relatively good measure for assessing one's own productivity [Meyer et al., 2014], the variance in self-reported tasks, especially also across developers, suggests that it might be a somewhat individual help for assessment at best.

2.5.2 How Fragmented is the Work?

To answer our second research question, "How fragmented is a developer's work?", we analyze how much time they spend on an activity before switching to the next one. The last three columns of Table 2.4 present the times a developer pursues each one of the activities before switching to another one. With the exception of planned meetings, a developer only remains in an activity between 0.3 (±2.6) and 2.0 (±6.5) minutes before switching to another one. These very short times per activity and the variety of activities a developer pursues each day illustrate the high fragmentation of a developer's work. The low standard deviations for each activity further emphasize this finding, which is similar to previous observations by González and Mark [2004], Ko et al. [2007], and ourselves [Meyer et al., 2014]. At the same time, our data also suggests that there are exceptions to this high work fragmentation and that in rare occasions, developers spend long hours without switching activities. For example, participant S4 was coding in the late

afternoon for 135.7 minutes, without any break longer than 2 minutes. Planned Meetings are the only exception to the short time periods spent on a single activity with an average duration of 15.8 minutes (±35.3) before switching. Our analysis of the data also suggests that developers are not using their computer in most of these planned meetings. The opposite is true for informal meetings, for which our monitoring tool recorded user input every few minutes. It is important to note that an activity switch, while contributing to work fragmentation, is not necessarily a task switch. A developer might switch activities several times while working on the same task, e.g. switching from coding in the IDE to the web browser to search for API documentation or code snippets.

To better understand the high number of activity switches, we performed an n-gram analysis to identify the activities which developers often perform in a sequence together. The activity pattern, which occurred most often, was a quick switch to emails during coding tasks. This finding is supported by our results in a previous observational study, where we learnt that developers often perform very quick and short context switches during waiting times, which increases their perceived productivity [Meyer et al., 2014]. Similarly, Amann et al. [2016] found that developers continue working while builds run in the background. Developers also regularly switch away from coding to work related web browsing (22.1%), reading or writing documents (14.3%) or planning (14.2%). These switches can be explained with the search for additional information necessary to complete a task, such as a task description from an email, a quick research on the web (*e.g.*, for a code snippet or tutorial), or reading a documentation-file. After these quick switches, developers usually switch back to their main coding task.

We were also interested in better understanding what activities developers were performing before they were interrupted by a co-worker, to learn where to focus for building task resumption tools. When developers switched their activity to an informal meeting, they were emailing in 40.1% of the cases, coding in 18.1% of the cases, and browsing the web (work related) in 13.5% of the cases before the switch. Switches to work unrelated web browsing were most often caused during coding tasks (35.5%) and during work related web searches (26.7%), likely to get a quick break from work.

2.5.3 Perceived Productivity Changes?

Our third research question asks "Are there observable trends in how developers perceive productivity?". We use the developers' self-ratings of their productivity via the pop-up survey to investigate this question. For each participant, we plot the perceived productivity ratings against the time of day the rating was collected; thus, all ratings from an individual are condensed across the hours of the day in one plot.

From an analysis of these plots, we found that although there was a lot of variation between individuals, the plots can be categorized into three broad groups: morning people, afternoon people, and those whose perceived productivity dipped around lunch. Figure 2.4 shows examples of these three types. The curved regression line in the figures shows the overall pattern of what part of the day an individual developer typically felt more or less productive with the shaded area showing the confidence range. Morning people were rare in our sample set (20% of all participants); Figure 2.4(a) shows S5's perceived productivity pattern, which is our clearest example of the trend but is not very pronounced. Afternoon people (8 out of 20, 40%) may be those who are industrious later in the day, or that feel more productive as a result of having the majority of their work day behind them (Figure 2.4(c)). The greater number of afternoon people in our sample reflected previous research that showed that information workers perceive themselves as most productive in mid-afternoon, peaking around 2-3 P.M. [Mark et al., 2014]. The low-at-lunch group (35%) may see long breaks as unproductive, or they may simply lag in effectiveness as their physical processes draw focus away from work (Figure 2.4(b)).

These graphs and numbers suggest that while information workers in general have diverse perceived productivity patterns, individuals do appear to follow their own habitual patterns each day. Only for one of the twenty participants it was not possible to determine a dominant category.

2.5.4 What are Productive Activities?

To answer our fourth research question, "What is the relationship between developers' activity and perceived productivity at work?", we built explanatory models relating the action and activity data to the productivity ratings.

The purpose of the **explanatory models** is to describe which factors contribute to the productivity ratings reported by the study participants. For each participant, we built one stepwise linear regression model for a total of 20 models. We chose linear regression because it is a simple and intuitive way to model data. The dependent variable is the reported productivity rating and the independent explaining variables are: (1) session duration, (2) number of certain events such as activity switches, (3) keystrokes per minute, (4) mouse clicks per minute, (5) amount of time spent in activities normalized by session length, and (6) how much of a session was before mid-day (noon) in percentage[2]. By choosing linear regression, we assume that the productivity ratings are interval data meaning that the distance between the productivity ratings 1 and 2 is the same as the distance between the ratings 2 and 3, and so on. To facilitate comparison across models, we specified the intercept value for all models as 4, which corresponds to an average perceived productivity.

Table 2.5 shows the results of the explanatory modeling. Each column corresponds to the perceived productivity model of a participant and each row corresponds to a factor in the model. To reduce the complexity of the table, we only report the sign of the coefficients; the full coefficients are available as supplemental material [3]. A plus sign (+) in a cell indicates that a factor has positive influence in a model; for instance, S1 reported higher productivity ratings with a higher number of self-reported tasks. Similarly, a minus sign (–) indicates negative influence; for instance, S3 reported lower productivity ratings for higher session durations. Empty cells correspond to variables that either were removed as part of the stepwise regression, or were not statistically significant. An NA value indicates that an event or activity did not occur for a participant

[2]We chose mid-day, since a previous study found differences in knowledge workers' activities before and after mid-day [Mark et al., 2014].

[3]`https://www.ifi.uzh.ch/en/seal/people/meyer/personal-analytics.html`

in the study period; for instance, the NA for S1 in *Debugger Use* means that S1 never used the debugger in the IDE during the study period. In a few cases, we were also not able to map all 'idle' log entries to the two activity categories of informal meetings or planned meetings due to a lack of information provided in the self-reports. These cases are also denoted with NA.

Based on the results presented in the table we can make several observations:

(i) No two explanatory models are the same. This suggests that productivity is a highly personal matter and that perceptions of what is considered to be productive are different across participants.

(ii) No single factor provides explanatory power across all participants. Furthermore, the same factor can have positive influence for one participant, and negative influence for another participant, for example the Number of Self Reported Tasks has both negative (2×) and positive influence (2×).

(iii) The Number of Keystrokes and the Number of Mouse Clicks have more often positive influence (7×) than negative influence (1× and 2× respectively).

(iv) The activities Email (5× negative), Planned Meeting (6× negative), Work Unrelated Browsing (5× negative), and Idle (6× negative) have more often negative influence.

Table 2.5: Explanatory Productivity Models for Participants ('+' indicates positive, '-' negative influence; 'NA' indicates a never performed activity. Columns 'Neg' / 'Pos' count the number of times a variable had negative / positive influence. The ratings are distributed on a 7-point Likert scale: left = "not at all productive" (1), right = "very much productive" (7)).

Neg	Pos	NA	Participant	S1	S2	S3	S4	S5	S6	S7	S8	S9	S10	S11	S12	S13	S14	S15	S16	S17	S18	S19	S20
			Ratings (total)	45	101	62	94	40	62	80	73	89	92	51	40	76	88	71	42	62	53	100	30
			Ratings (discarded)	0	29	3	4	0	0	0	7	10	10	1	0	7	0	0	0	4	8	11	4
			Ratings (included in model)	45	72	59	90	40	62	80	66	79	82	50	40	69	88	71	42	58	45	89	26
			Ratings (distribution)																				
3	3	0	**Session Duration (in hours)**		+	−	−		+		−					+							
2	1	0	**Percent of Session Before Noon**											−			−				+		
			Per Minute																				
2	2	0	# self-reported tasks	+								−		−		+							
2	4	0	# activity switches		+		−		+				+	−				+					
1	1	4	# meetings	NA		NA				+				NA				NA	−				
2	1	7	# instant messaging switches	NA	NA	NA	NA	+		NA	NA		NA								+		−
1	**7**	0	# keystrokes					+						+		+	−	+		+	+		+
2	**7**	0	# mouse clicks	+		+				+		+	+	+	−		+				−		
			Percent Activity																				
4	1	0	Dev. Coding						−				−			−					+		−
0	1	9	Dev. Debugger Use	NA	NA	NA				NA	NA	NA	NA	+							NA		NA
2	1	12	Dev. Code Reviews	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	−			NA	NA				−	+
2	0	3	Dev. Version Control		NA		−		NA				−							NA			
5	0	0	Email							−			−	−		−						−	
2	3	0	Planning		+		+							−	+							−	
4	3	0	Read / write documents	+		+			−				−	+		−				−			
6	0	5	Planned meeting	NA		NA		−	−	−		−		NA	NA	−		NA					−
3	2	9	Informal meeting	NA	+	NA	NA	NA					−	NA	NA	−	NA	NA		NA	+		−
2	0	8	Instant messaging	NA	NA	NA	NA	NA	−	NA	NA		NA								−		
1	2	0	Work related browsing							−			+								+		
5	0	2	Work unrelated browsing					−			NA		−			−	−	−		NA			
2	1	0	Other								+						−						−
2	1	4	Other RDP	NA		NA		−			+						NA	−					NA
6	2	2	Idle		−		−	−	−		+	−		NA		−	+	NA					

2.5.5 Summary of Results

Our analysis provides a broad range of insights on the relationship between developer's work practices and activities and their perceived productivity. Table 7.4 summarizes some of the key findings.

Table 2.6: Summary of Some of the Study Key Findings.

#	Finding	Section
F1	Developers only spend about half their time active on their computer.	2.5.1
F2	For every work hour, developers have an average of 2.5 short breaks, totaling 10.5 minutes of unplanned time away from their computer.	2.5.1
F3	Developers spend about a fourth of their time on coding related activities and another fourth of their time on collaborative activities.	2.5.1
F4	The range and time spent on activities varies greatly depending on the individual and company.	2.5.1
F5	Developers' work is highly fragmented, spending very short amounts of time (0.3 to 2 minutes) in one activity before switching to another one.	2.5.2
F6	Developers' perceived productivity follows habitual patterns, broadly categorisable as morning people, afternoon people and "low-at-lunch" people.	2.5.3
F7	Productivity and the factors that influence it are highly individual.	2.5.4
F8	The number of mouse clicks and key strokes often have a more positive, email, planned meetings, and work unrelated browsing a more negative impact on perceived productivity.	2.5.4

2.6 Threats to Validity

The main threats to our study are construct validity threats due to the monitoring application we used to collect data. These, and the threats to internal and external validity, are described in this section.

2.6.1 Construct Validity

The main threat comes with the metrics we base our analysis on, as it is limited to the data we collected with our monitoring application. We believe that a chronological record of application use, user inputs, and the addition of self-reported productivity, tasks, and activities provide a reasonable basis to analyze a developer's work. Though, we cannot exclude the possibility that any other factors influence a developer's work day and productivity.

Running a monitoring application in a real-world scenario might capture inaccurate data, due to bugs in the logging application or stability issues. To mitigate this risk, we ran several test-runs in different scenarios prior to the study, and observed a user for several hours to compare the logged with the observed data. No major problems with the tracker were reported during the tests or at the time of the study.

Even though the monitoring tool is able to capture a broad range of activities on a participants' computer, it does not capture activities away from the computer. Therefore, we asked participants to record their activities/tasks for their time away from the computer in the periodic self-reports. Furthermore, to capture activities performed on secondary computers or remote desktops, we asked participants to install the monitoring application without the self-reporting feature on these machines as well.

Understanding, categorizing and analyzing the data poses another threat to validity, especially since it is not straightforward to identify all activities from the collected data. For instance, mapping 'idle' times to self-reported breaks, planned meetings and informal meetings could not be automated. We also needed to discard outliers, such as very short work in the middle of the night or on weekends. To mitigate this risk, we did a manual mapping of activities we were

uncertain about and checked random samples of the semi-automated mapping. Assumptions made and thresholds defined were carefully discussed, based on previous related work, and described in the paper in detail. The short interviews at the end of the study further helped us to interpret each participants' data and work behavior.

2.6.2 Internal Validity

Running a monitoring application on participants' computers might pose privacy concerns to participants. To address these concerns, we tried to be very transparent about the data we collected. The participant was shown the location where the logs were saved and given the opportunity to censor them. We did not collect information about what was being typed or clicked on, merely that these events were occurring. We also assured the participant that all collected data will be anonymized and saved on password-protected devices or in locked filing cabinets.

Monitoring participants' during their work bears the risk of changing their behavior. To mitigate these risks, we tried to collect as much data as possible in the background and optimized the performance of the data collection to avoid lags, creating a non-intrusive experience for participants. Several participants explicitly mentioned that they usually forgot about the monitoring application and were only reminded when they were prompted about the self-reports.

Interrupting participants with a short survey once an hour might have influenced their work behavior and habits. To address these concerns, we tried to only show the pop-ups when necessary and reduce the effort needed to fill them out by showing previous responses, having quick response buttons, and auto-completion boxes. Additionally, the participant had the chance to postpone the survey in case it interrupted at an inopportune moment. The continuously very short but stable amount of time used to answer the periodic survey throughout the study and the small variation in the number of responses per participant and day also suggests that participants' behavior was not affected much and that they did not get annoyed by the survey.

2.6.3 External Validity

The number of participants or the selection of participants might limit the generalizability of the results of this study. In particular, our participants were all using Windows as their operating system due to our monitoring application being built for Windows. Overall though, we tried to mitigate the threats to external validity and generalizability by selecting participants from four different software companies of varying size, with more and less well-established products, different kinds of customers, and different stages of product development. Studies were spread to three different countries, Canada, US, and Switzerland, and across half a year. Additionally, all participants are professionals who were studied in their everyday, real-world work environment and not in an experimental exercise. The external validity is further strengthened by the broad range of development experience of our participants, ranging from junior to senior developers with an average professional development experience of 14.2 years ($\pm$9.6, ranging from 0.5 to 40 years). Finally, our participants worked on projects using 7 of the top 10 most used programming languages according to a recent large-scale study [StackOverflow, 2017]. While the large-scale study also showed that 55% of all developers use Windows as an operating system and our focus on Windows thus maps to a majority of developers, further studies with a broader participant pool are needed to assess the generalizability of our results.

Another limitation might be the study running for roughly two weeks per participant, as developers' activities might vary greatly at different stages and iterations of their projects. We tried to mitigate this risk by having participants from different teams at different stages of their project and iterations, and by staggering the monitoring period between participants so that varying times of the year were covered. In the final interview, most participants also agreed that the study took part in fairly usual, and not extraordinary, work weeks.

2.7 Discussion

The results of our study shed new light on the work practices, fragmentation and perceptions of productivity of individual software developers. In the following, we discuss implications and opportunities of our findings, in particular the individuality of productivity and its use for designing better tool support, and we report on an exploratory analysis to predict high and low productivity sessions.

2.7.1 Individuality of Productivity

To quantify a developer's productivity, related work predominantly focused on a single or a small set of outcome measures, such as the lines of code or function points. While these measures can be used across developers, they neglect to capture the individual differences in the way that developers' work as well as the differences in their work and their perceived productivity. The results of our study show that perceived productivity is in fact a very personal matter and the influence and impact of the factors used for our explanatory productivity models varied greatly. This suggests that measures or models of productivity should take into account the individual differences in what is perceived as productive or not and capture a developer's work more holistically rather than just by a single outcome measure. Such individual models could then be used to provide better and more tailored support to developers, for instance, to foster focus and flow at work.

At the same time, our results also show that while there are individual differences, there are tendencies amongst groups of software developers, for instance, with the number of key strokes and mouse clicks having a positive influence on productivity perception for 7 of the 20 participants. Similarly, we identified types of developers with similar trends of perceived productivity over the course of the day, including morning and afternoon people, which resemble the morningness and eveningness types that Taillard et al. [1999] identified in their large-scale study on people's sleeping habits. These results suggest that it might be possible to identify clusters of software developers with fairly similar productivity models despite individual differences, which could then be used

to provide tool support tailored to these clusters, for instance, for scheduling a productive work day.

2.7.2 Supporting Flow and Retrospection

In our previous study, we found that developers feel particularly productive when they get into "the flow" without having many switches [Meyer et al., 2014]. Results from this and other studies suggest that getting into the flow during work might not be very easy, given the high fragmentation of work and the many short breaks and interruptions.

At the same time, our analysis of the collected data suggests that it might be possible to identify individualized proxies for developer productivity, such as using the number of mouse clicks or key strokes per minute or the time spent in work-unrelated browsing for certain developers. Knowing if a developer is productive or unproductive at the moment by using such proxies could be used to support getting and staying in a highly productive "flow" state. In particular, one could use this to indicate the availability of a developer for interruptions by changing the availability status in instant messaging tools, or with a visual external cue to avoid external interruptions at particularly inopportune moments, similar to what Züger and Fritz [2015] suggested. Or, one could also use this to provide awareness to the developers themselves on their flow and productivity, by indicating them when they are stuck and it might be time to ask a co-worker for help or to take a break, or even blocking work-unrelated websites for 25 minutes—similar to the PomodoroTechnique [2019]—and helping them to focus when they are procrastinating.

Being able to predict, to some extent, a developer's productivity on an individual basis could also be used to provide developers with individualized retrospection support, in the form of daily or weekly summaries of their work. With the *Quantified Self* movement, more and more people are using applications and devices to track themselves—mostly with a focus on the non-work related activities, such as sports. These persuasive technologies provide users an opportunity to reflect upon their own activities and support desired improvements, such as a more active lifestyle (*e.g.*, [Bravata et al., 2007; Consolvo et al., 2008b;

Fritz et al., 2014]), due to self-monitoring and goal-setting [Fritz et al., 2014]. A proxy of an individual developer's productivity might provide such benefits for the software development domain, by increasing developers' awareness about their own work practices and productivity and thereby helping them to improve them.

2.7.3 Scheduling a Productive Work Day

By knowing the trends of developers' perceived productivity and the activities they perceive as particular productive/unproductive, it might be possible to schedule the tasks and activities developers must perform in a way that best fits their work patterns. For example, if a developer is a morning person and considers coding particularly productive and meetings as impeding productivity, blocking calendar time in the morning for coding tasks and automatically assigning afternoon hours for meeting requests may allow the developer to best employ their capabilities over the whole day. Or, it could remind developers to reserve slots for unplanned work or interruptions at times where they usually happen.

2.7.4 Predicting High & Low Productivity

To examine whether we might be able to identify high and low productivity sessions with the collected data, we performed an initial, exploratory analysis, building **predictive models** using logistic regression. For each participant, we computed the median productivity rating individually, which we assumed to be the standard perceived productivity of a developer. We then used the productivity (high and low) as the dependent variable and the factors used in the explanatory models (see Table 2.5) as the independent variables, and we built two prediction models using binary logistic regression:

Model 1: Is the reported perceived productivity *above* the median productivity? (High Productivity)

Model 2: Is the reported perceived productivity *below* the median productivity? (Low Productivity)

Table 2.7: Models to Predict High / Low Productivity Sessions.

	Ratings	High Productivity			Low Productivity		
Partic.	Distr.	Ratio	Prec.	Recall	Ratio	Prec.	Recall
S1		0.36	0.64	0.39	0.24	0.85	0.03
S2		0.54	0.76	0.71	0.15	0.41	0.46
S3		0.36	0.69	0.48	0.39	0.54	0.58
S4		0.13	0.46	0.11	0.39	0.62	0.33
S5		0.13	0.55	0.27	0.45	0.53	0.47
S6		0.27	0.32	0.43	0.26	0.56	0.59
S7		0.38	0.63	0.64	0.26	0.36	0.11
S8		0.03	0.60	0.42	0.02	0.52	0.68
S9		0.41	0.41	0.43	0.27	0.62	0.38
S10		0.07	0.17	0.41	0.09	0.56	0.48
S11		0.36	0.60	0.62	0.26	0.48	0.42
S12		0.50	0.71	0.76	0.50	0.67	0.62
S13		0.41	0.59	0.62	0.45	0.54	0.52
S14		0.49	0.60	0.64	0.27	0.59	0.45
S15		0.46	0.67	0.77	0.18	0.64	0.18
S16		0.29	0.54	0.40	0.40	0.73	0.52
S17		0.34	0.61	0.52	0.22	0.29	0.13
S18		0.18	0.47	0.40	0.47	0.54	0.49
S19		0.25	0.72	0.08	0.43	0.48	0.54
S20		0.50	0.67	0.52	0.50	0.60	0.62
Average			**0.57**	**0.48**		**0.56**	**0.43**

We built and evaluated the models for each participant using 50 random split experiments: 2/3 of the participant's data was used for training and the remaining 1/3 of the data was used for testing the model. In total, we ran $50 \times 2 \times 20 + 50 = 2\,050$ experiments. For each experiment, we measured the success of the predictions with precision and recall. Precision represents how many of the returned results are relevant (correct), and recall represents how many of the relevant results were returned. We then averaged the precision ratings over the 50 experiments for each model and participant to receive a single precision rating. We did the same for recall.

Table 2.7 shows the results. In addition to the precision and recall values, we report in the Ratio columns the percentage of High productivity and Low productivity sessions for each participant. On average, the models have a precision of 0.57 (High Productivity) and 0.56 (Low) and recall of 0.48 (High) and 0.43 (Low). For some participants, the precision and recall values are above 0.70. The results are promising and suggest that even with a relatively small number of reported productivity ratings, it is possible to build personalized, predictive productivity models. To build the models we used the session length and information about events (keystrokes, mouse clicks), and activities. We expect that with more context and data, such as active task information, window contents, calendar information, development experience, time of day, or possibly biometrics, the quality of the predictions can be improved further.

2.7.5 Privacy Concerns

As with all approaches that collect personalized data on people, collecting information on a developer's activities on the computer potentially raises many privacy concerns. Especially given the focus of our study on productivity, some people were skeptical and declined to participate. This indicates the sensitivity of the data and the need for further research on the privacy concerns of such broad, work-related data. We believe that integrating potential users early on in the design process of building such a tool is crucial to increase acceptance of and engagement with the tool. Furthermore, we expect that the voluntary use of such applications and its ability to tailor to the individual is important for its success since it focuses on the intrinsic motivation of developers to improve or better understand themselves. Requiring the use of such tools by upper management on the other hand will lead to a 'gaming' as previous research found and suggested (*e.g.*, [Treude et al., 2015]), since developers might fear that the gathered information could influence their employment and increase pressure.

2.8 Summary

Related work has proposed a wide variety of approaches to quantify the productivity of software developers, mostly only taking a single aspect of a developers' work into account. In our paper, we present a more holistic approach to examine developers' work practices and their relation to perceived productivity. We conducted a study with 20 professional software developers in four different companies *in situ*, to investigate how developers spend their work days and what activities they perform. We applied a combination of computer logging and experience sampling by using a background monitoring application and by letting developers answer pop-up questions every 60 minutes. The results show that developers spend their time on a wide variety of activities, about a fourth of it on collaborative activities, such as meetings or emails. On average, participants spend only between 0.3 and 2 minutes on each activity, before switching to the next one. They also have 2.5 short per hour and 3.3 breaks per day. This demonstrates how fragmented a developer's normal work day is.

Based on developers' self-reports, we analyzed how their perceived productivity changes throughout a work day. We found a lot of variation between individuals, but that they can roughly be grouped into morning people, low-at-lunch people and afternoon people. We also correlated perceived productivity with activities and user input using a stepwise linear regression model per participant. The data suggested that productivity is a personal matter and that perceptions vary greatly as different factors in a developer's work day can influence productivity either positively or negatively. More user input was often associated with a positive, while emails, planned meetings and work unrelated websites were most often associated with a negative perception of productivity. Based on our findings, we propose a number of design approaches and tools to help increase developer productivity. For instance, by supporting developers to get into and stay in "the flow", by reducing interruptions at inopportune moments and by helping them to focus when they are procrastinating. Finally, we ran an exploratory analysis of predicting productivity for individuals, based on their computer usage. The results are promising and suggest that even with

a relatively small number of reported productivity ratings, it is possible to build personalized, predictive productivity models. In the future, we plan to work on improving the quality of these predictions by including more context and data, such as active task information, experience, time of the day, and biometrics.

2.9 Acknowledgments

The authors would like to thank the study participants and all of our reviewers for their insightful remarks. This work was funded in part by ABB, SNF and NSERC.

3

Characterizing Software Developers by Perceptions of Productivity

André N. Meyer, *Thomas Zimmermann, Thomas Fritz*
Published at the 2017 ESEM Conference on Empirical Software Engineering and Measurement,
Contribution: Study design and execution, participant recruitment, data collection, partial data analysis, and paper writing

Abstract

Understanding developer productivity is important to deliver software on time and at reasonable cost. Yet, there are numerous definitions of productivity and, as previous research found, productivity means different things to different

developers. In this paper, we analyze the variation in productivity perceptions based on an online survey with 413 professional software developers at Microsoft. Through a cluster analysis, we identify and describe six groups of developers with similar perceptions of productivity: social, lone, focused, balanced, leading, and goal-oriented developers. We argue why personalized recommendations for improving software developers' work is important and discuss design implications of these clusters for tools to support developers' productivity.

3.1 Introduction

Understanding, measuring and optimizing software developers' productivity is important to deliver software on time and at reasonable cost and quality. Previous work introduced numerous measures of productivity that vary by their focus on specific outputs of development work, such as lines of code [Devanbu et al., 1996; Walston and Felix, 1977], function points [Jones, 1994], or completed tasks [Zhou and Mockus, 2010], over time. Other researchers have looked at organizational factors and their impact on developer productivity, such as the team [Blackburn et al., 1996; Boehm et al., 2000; Melo et al., 2011] and workplace characteristics [DeMarco and Lister, 2013]. While these measures and factors can be valuable to compare certain aspects of productivity, they neglect to capture the many differences in perceptions of productivity as well as the differences in developers' work, roles and habits. By investigating developers' perceptions of productivity, several researchers concluded that developers are different in what they consider as productive or unproductive [Johnson et al., 2003; Melo et al., 2011; Meyer et al., 2014; Vasilescu et al., 2016b]. Yet, little is known about the characteristics, the variation, and the commonalities in developers' productivity perceptions. A better understanding of these aspects of developers' productivity perceptions can help to provide better and more tailored support to developers. In this paper, we explore the characteristics of developers' perceptions of productivity and the clusters of developers with similar perceptions. We report on the results from an online survey with 413 professional software developers at Microsoft. We show that developers can roughly be clustered into six groups with

similar perceptions—social, lone, focused, balanced, leading, and goal-oriented developer—thus allowing to abstract and simplify the variety of individual productivity perceptions. We characterize these groups based on the aspects that developers perceive to influence their productivity positively or negatively, and by the measures developers are interested in to reflect about their productivity. We discuss the implications of our clusters on software development and their potential in optimizing developers' productivity and support tools.

3.2 Related Work

To avoid repetitions in this thesis, we present and summarize related work on developer productivity and factors that influence it in Section 1.6 of the synopsis.

3.3 Methodology

To explore developers' productivity perceptions, in particular the variations and similarities amongst developers, we designed and conducted a survey and analyzed the collected answers from 413 participants.

3.3.1 Data Collection

We conducted an online survey, consisting of four main questions about productivity perceptions, at Microsoft.

Survey Design. The first two questions **Q1** and **Q2** asked the participants to describe a productive and an unproductive work day in two words each ("Please describe what a *productive* work day is to you in two words.", "Please describe what an *unproductive* work day is to you in two words."). We prompted users for two keywords each to foster more than one precise definition of productivity, similar to what we applied in a previous study [Meyer et al., 2014]. The third question **Q3** asked the agreement with statements on factors that might affect productivity. The last question **Q4** asked about the interestingness of productivity measures

at work. The order of the questions was chosen to not bias the participants when they described productive and unproductive work days (Q1 and Q2), before showing them the list of statements and measures (Q3 and Q4). The complete survey can be found as supplementary material . None of the questions were required to be answered and participants could stop the survey at any point in time.

Productivity perceptions (Q3). For question Q3, "Please rate your agreement with each of the following statements", we used a symmetric, five-point Likert scale from *strongly agree* (5) to *strongly disagree* (1) to ask about the agreement with 20 statements on when people feel productive, for example, "*I feel productive when I write code.*" The statements were selected from related work that analyzed the impact of various work patterns on productivity. We focused on statements about activities software developers pursue during a work day and the fragmentation of their work. Specifically, we asked participants about the perceived relation between productivity and coding related activities (that is, their main work activity) [Albrecht, 1979; Meyer et al., 2014]; social activities such as emails [Mark et al., 2016b], meetings [Meyer et al., 2014], and helping co-workers [Chong and Siino, 2006; Meyer et al., 2014]; work unrelated activities such as breaks [Epstein et al., 2016a]; the fragmentation of their work such as the impact of distractions and multi-tasking [DeMarco and Lister, 2013; González and Mark, 2004; Mark et al., 2016b]; the time of the day [Mark et al., 2014; Sach et al., 2011; Spira and Feintuch, 2005]; and their happiness at work [Graziotin et al., 2014a; Khan et al., 2011].

Productivity measures (Q4). For question Q4, "Please rate how interesting each of the following items would be for you to reflect on your work day or work week", we used a symmetric, five-point Likert scale from *extremely interesting* (5) to *not at all interesting* (1) to ask about the interestingness of 30 potential measures of productivity to reflect about work, such as "The time I spent coding." or "The number of emails I sent." The measures were selected as follows: for the categories, which we identified for Q3 from related work, we selected measures

related to how much time was spent on an activity/event and the total number of times an activity/event occurred. We further added measures related to the overall time spent on the computer and within various applications, and the tasks worked on, which developers in another study on productivity considered to be most relevant [Meyer et al., 2014].

Participants. We advertised the survey by sending personalized invitation emails to 1600 professional software developers within Microsoft. To incentivize participation, we held a raffle for two US$ 50 gift certificates. In total, 413 people participated in the survey (response rate of 25.8%); 91.5% of the participants reported their role to be individual contributor, 6.8% team lead or manager, and 1.7% stated they are neither. Participants had an average of 9.6 years (±7.5, ranging from 0.3 to 36) of professional software development experience.

3.3.2 Data Analysis

We used the responses to Q3 to group participants with similar perceptions of productivity together. First, we normalized the responses. When responding to surveys, some participants are more positive than others, which can lead to biases in the responses. For example, Alice might center her responses to the question Q3 around the response to "agree", while Bob tends to center his responses around the response "neutral". To correct for such personal tendencies, we normalized responses to Q3 and Q4 as follows. We treated the scale as numeric and for each survey participant we computed the median response for Q3 and Q4 respectively: $median_{Q3}$ and $median_{Q4}$. We then subtracted the median from the responses and computed the sign. More formally, for the response $r_{q,I}$ to a question q and item I, we normalize with sign($r_{q,I}$ – $median_q$). As a result, we end up with three categories: A value of +1 indicates that a participant responded more positively about an item than for most of the other items (HIGHER). A value of –1 indicates that a participant was more negative about an item (LOWER). A value of 0 indicates that a participant was neutral towards an item. We used medians instead of means because they more effectively capture neutral responses as zero. Next, we clustered participants into groups using

the *pamk* function from the *fpc* package in R. The input was the normalized responses to Q3. The *pamk* function is a wrapper function to the commonly used *pam* clustering function. The wrapper computes the optimal number of clusters. In our case, the optimal number of clusters was six. The resulting clusters — social, lone, focused, balanced, leading, and goal-oriented developers —are discussed in the next section. Finally, to describe the inferred groups, we used the responses to questions Q1, Q2, and Q3. We created comparison word clouds for the responses to Q1 and Q2 using the *wordcloud* package in R. These word clouds depict the relative frequency of the most frequently used words for each cluster, with more frequently used words being displayed in a bigger font size. Furthermore, we used the responses to Q4 to identify the measures that developers of a cluster are interested in.

3.4 Results

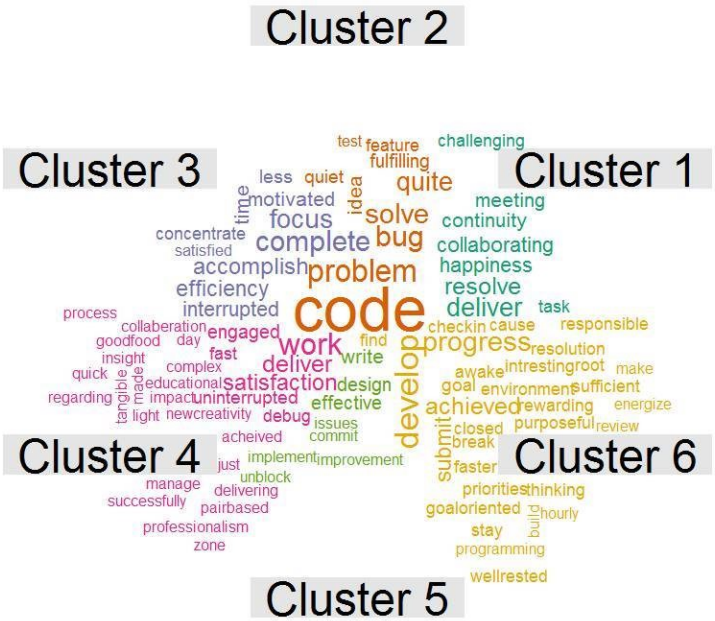
We identified the following six clusters based on our analysis of Q3. We further describe them based on the responses from Q1 (see the word clouds in Figure 3.1, one color per cluster), Q2 (Figure 3.2) and Q4:

1. The *social developers* (C1) feel productive when helping coworkers, collaborating and doing code reviews. To get things done, they come early to work or work late and try to focus on a single task.

2. The *lone developers* (C2) avoid disruptions such as noise, email, meetings, and code reviews. They feel most productive when they have little to no social interactions and when they can work on solving problems, fixing bugs or coding features in quiet and without interruptions. To reflect about work, they are mostly interested in knowing the frequency and duration of interruptions they encountered.

3. The *focused developers* (C3) feel most productive when they are working efficiently and concentrated on a single task at a time. They are feeling unproductive when they are wasting time and spend too much time on

Figure 3.1: Comparing the clusters with respect to words that developers associate with ***productive*** (Q1) work days. Terms in **turquoise** are related to Cluster 1, **orange** to Cluster 2, **purple** to Cluster 3, **pink** to Cluster 4, **green** to Cluster 5, and **gold** to Cluster 6. The size of a term corresponds to the difference between the maximum relative frequency and the average relative frequency of the word across the s

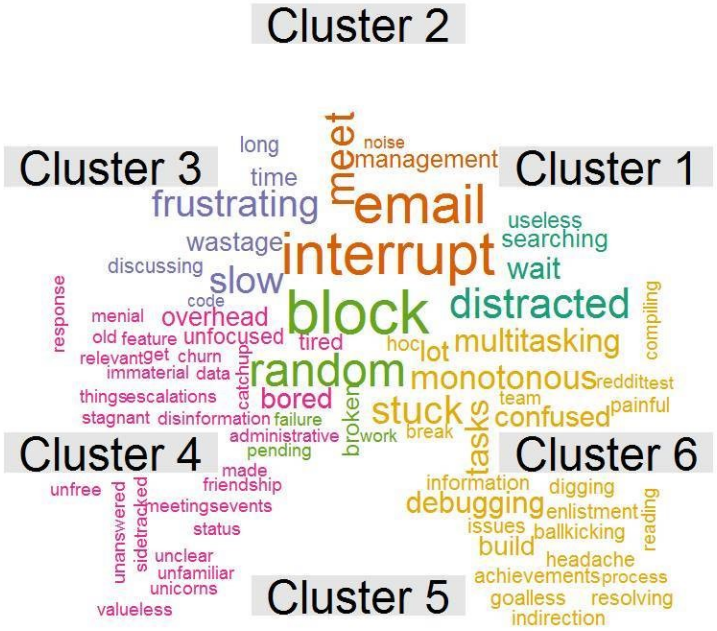


a task, because they are stuck or working slowly. They are interested in knowing the number of interruptions and focused time.

4. The *balanced developers* (C4) are less affected by disruptions. They are less likely to come early to work or work late. They are feeling unproductive, when tasks are unclear or irrelevant, they are unfamiliar with a task, or when tasks are causing overhead.

5. The *leading developers* (C5) are more comfortable with meetings and emails and feel less productive with coding activities than other developers. They feel more productive in the afternoon and when they can write and design things. They don't like broken builds and blocking tasks, preventing them (or the team) from doing productive work.

Figure 3.2: Comparing the clusters with respect to words that developers associate with ***unproductive*** (Q2) work days. The same color coding and text size applies as in Figure 3.1.



6. The *goal-oriented developers* (C6) feel productive when they complete or make progress on tasks. They feel less productive when they multi-task, are goal-less or are stuck. They are more open to meetings and emails compared to the other clusters, in case they help them achieve their goals.

Table 3.1 describes the characteristics of the clusters in more detail, in particular the cluster name, the statements for which half or more participants in the cluster gave HIGHER scores for Q3 (second column), and the statements for which half or more participants in the cluster gave LOWER scores for Q3 (third column). Prefixed with 🔨, the table also lists the productivity measures (from question Q4) which were interesting (second column) or not interesting (third column) to the majority of developers within a cluster. The tendency reported in the table corresponds to the average normalized score. If the tendency is not reported for a statement, it means it was greater than 0.500 (second column)

or lower than –0.500 (third column). The first row lists the statements that were scored higher/ lower by most participants (50% or more) in the majority of clusters (four or more). As an example, the statement "I feel productive when I write code" was scored higher by more than 50% of people in clusters C1, C2, C3, C4, and C6. This was not the case for cluster C5, which is reported as an exception, both in the first row and the row corresponding to C5. Other statements scored higher by most developers in most clusters are "I feel productive on a day with little to no meetings", "I feel productive when I am happy", and "I feel productive when I have fewer interruptions".

Table 3.1 also shows that some measurements (🔨) are of interest for reflection on work (Q4) to most clusters. People in most clusters gave higher scores to the time spent coding and the longest period focused on a task without an interruption. The number of open applications and the inbox size received lower scores overall. However, the table also highlights differences between the clusters with respect to the measurements that participants consider to be interesting. For example, the *lone developers* (C2) are interested in the number and duration of interruptions. They are less interested in the list of applications used and web sites visited. The *balanced developers* (C4) are interested in the tasks, the number of interruptions, and the focus over time. They are less interested in the number of emails sent and received. Several clusters are further related to each other along specific aspects. For example, C1 and C2 are related in how they perceive the productivity of social interactions. While social developers (C1) embrace them, *lone developers* (C2) feel more productive when having uninterrupted work alone. Further, clusters C3 and C6 are related, as *focused developers* (C3) are more interested about working efficiently, while *goal-oriented developers* (C6) feel the most productive when they get work done.

Table 3.1: The Six Clusters/Personas from the Survey

	50% OR MORE GIVE **HIGHER** SCORES FOR (TENDENCY: > 0.500)	50% OR MORE GIVE **LOWER** SCORES FOR (TENDENCY: < –0.500)
Most clusters (four or more)	I feel productive when I write code (except C5) ● I feel productive on a day with little to no meetings (except C5 and C6) ● I feel productive when I am happy (except C2 and C5) ● I feel productive when I have fewer interruptions (except C5 and C6) 🔨 The time I spent coding (except C1) 🔨 The longest period focused on a task without an interruption	I feel productive when I send more emails than usual (except C6) ● I feel I had a productive work day when my email inbox is emptier in the evening than in the morning (except C1 and C4) ● I feel productive when I visit social networks or news websites to do a quick break (except C1 and C4) ● If I have many program windows open on my screen, it decreases my perceived productivity ● I feel productive on a particular day of the week, e.g., on Wednesdays (except C5) ● I feel more productive in the morning than in the afternoon (except C3) ● I feel less productive after lunch compared to the rest of the day (except C3 and C6) 🔨 The number of open application windows (except C1 and C5) 🔨 The inbox size in the course of the day/week (except C2, C5)
Cluster C1: **The social developer** Size: 62	I feel productive when I test or debug my code ● I feel productive when I do code reviews ● I feel productive when I help my coworkers ● I come early to work/work late to get some focused work hours ● I feel productive when I work on one task at a time 🔨 Exception: The time I spent coding (tendency: 0.478)	Exception: I feel I had a productive work day when my email inbox is emptier in the evening than in the morning (tendency: –0.258) ● Exception: I feel productive when I visit social networks or news websites to do a quick break (tendency: –0.403) 🔨 Exception: The number of open application windows (tendency: –0.370)
Cluster C2: **The lone developer** Size: 64	I feel productive when I test or debug my code ● I feel productive when I read fewer emails than usual ● Background noise distracts me from my work ● Exception: I feel productive when I am happy (tendency: 0.203) 🔨 The number of interruptions I had 🔨 The duration of each interruption	I feel productive when I do code reviews 🔨 A list of applications I used 🔨 The websites I visited the most 🔨 Exception: The inbox size in the course of the day/week (tendency: –0.438)
Cluster C3: **The focused developer** Size: 54	I feel more productive in the morning than in the afternoon ● I feel productive when I work on one task at a time 🔨 The tasks I worked on 🔨 The number of interruptions I had 🔨 My focus over the course of the day week	I feel more productive in the afternoon than in the morning ● Exception: I feel less productive after lunch compared to the rest of the day (tendency –0.155) 🔨 The number of emails I received 🔨 The number of emails I sent
Cluster C4: **The balanced developer** Size: 50		I come early to work/work late to get some focused work hours ● Exception: I feel I had a productive work day when my email inbox is emptier in the evening than in the morning (tendency: –0.180) ● Exception: I feel productive when I visit social networks or news websites to do a quick break (tendency 0.000)
Cluster C5: **The leading developer** Size: 97	Exception: I feel productive when I write code (tendency: 0.309) ● Exception: I feel productive on a day with little to no meetings (tendency: –0.103) ● Exception: I feel productive when I am happy (tendency: 0.268) ● Exception: I feel productive when I have fewer interruptions (tendency: 0.247)	I feel more productive in the afternoon than in the morning ● Exception: I feel productive on a particular day of the week, e.g., on Wednesdays (tendency: –0.400) 🔨 Exception: The number of open application windows (tendency: –0.447) 🔨 Exception: The inbox size in the course of the day/week (tendency: –0.478)
Cluster C6: **The goal-oriented developer** Size: 38	I feel productive when I work on one task at a time ● Exception: I feel productive on a day with little to no meetings (tendency –0.079) ● Exception: I feel productive when I have fewer interruptions (tendency: 0.447)	Exception: I feel productive when I send more emails than usual (tendency 0.135) ● Exception: I feel less productive after lunch compared to the rest of the day (tendency: –0.211)

3.5 Discussion

Understanding how developers perceive productivity is important to better support them and foster productivity. The six clusters and their characteristics provide relevant insights into groups of developers with similar productivity perceptions that can be used to optimize the work and flow on the team and the individual level. The differences between software developers' preferred collaboration and work styles show that not all developers are alike, and that the cluster an individual or team belongs to could be a basis for tailoring actions for improving their work and productivity.

On the **team level**, it might, for instance, be most beneficial to provide a quiet, less interruption-prone office space to lone and *focused developers* (C2 and C3) and seat social developers (C1) in open offices. Similarly, a team might benefit from an increased awareness about each members' communication preferences, to reduce ad-hoc meetings for *lone* and *focused developers* (C2 and C3) or use more asynchronous communication where they can choose when to respond to an inquiry. The group of developers can be further beneficial for task assignment. For example, an exploration task for a new product that is rather open without clear goals and that requires a lot of discussion might be less suitable for a *goal-oriented* (C6), a *lone* (C2) or a *balanced developer* (C4).

On the **individual level**, developers might benefit from tailored user experiences and feature sets for software development tools. For instance, a tool to foster productive work and avoid interruptions could block emails and instant messaging notifications for the *lone developer* (C2) while they are coding, but allow them for the *social developer* (C1), similar to what was previously suggested [Agapie et al., 2016]. Similarly, the code review or build experience could be adjusted based on different productivity perceptions. In addition, the clusters could be used for advice tailored to specific groups of developers, *e.g.*,, recommend the *focused developer* (C3) to come to work early to have uninterrupted work time, or suggest the *balanced developer* (C4) to take a break to avoid boredom and tiredness [Epstein et al., 2016a]. The clusters can help to quantify the individual productivity of developers more accurately, by considering what matters most to

them and depending on their perceptions of productivity. For example, a *leading developer* (C5) is likely feeling much more productive after a day with multiple meetings spread over the day, compared to the *focused developer* (C3), who only has little time to focus on the tasks in-between these meetings.

Overall, the identified clusters and the aspects that differentiate these clusters, such as goal-orientation, single-task focus or socialness, are a first step towards a set of "productivity traits" of developers. Similar to the big five personality traits (OCEAN) [John and Srivastava, 1999] that help to understand other people's personality, the self-assessment along such productivity traits can provide useful information for understanding oneself or other developers and for optimizing the work individually as well as in teams.

3.6 Threats to Validity

External validity. Due to the selection of participants, as all work for the same company, the results might not generalize to other software development contexts. We tried to mitigate this threat by advertising the survey to professional software developers in different product teams within Microsoft, at different stages in their projects, and with varying amounts of experience; resulting in a more diverse set of participants. By providing the survey questions, we encourage other researchers and practitioners to replicate the study in other companies.

Construct validity. The selection of questions that we asked in the survey also impacts the results. For example, questions about different dimensions of productivity, might lead to a different clustering. We created the questions based on factors that we identified in related work and from our previous experience with surveying and interviewing developers about their perceptions and measuring of productivity [Meyer et al., 2014]. The choice of clustering algorithm and approach of using questions Q1 to Q3 to describe the inferred clusters might also have influenced the results. Future work is needed to analyze the robustness and completeness of the productivity statements and clusters.

3.7 Conclusion

Different to previous work that suggested numerous productivity measures and found that perceptions of productivity can vary greatly between developers, our research provides an exploratory first step into identifying commonalities and underlying categories of developers' productivity perceptions. Based on the clustering of developers' answers to productivity statements mentioned in related work, we identified and characterized an initial set of six such categories and discussed their potential to improve the work and flow of software developers and their teams.

3.8 Acknowledgements

The authors would like to thank the study participants.

4

Today was a Good Day: The Daily Life of Software Developers

André N. Meyer, *Earl T. Barr, Christian Bird, Thomas Zimmermann*
Accepted in the 2019 IEEE Transactions on Software Engineering Journal,
Contribution: Data analysis and paper writing

Abstract

What is a good workday for a software developer? What is a typical workday? We seek to answer these two questions to learn how to make good days typical. Concretely, answering these questions will help to optimize development processes and select tools that increase job satisfaction and productivity. Our work adds to a large body of research on how software developers spend their time. We report the results from 5971 responses of professional developers at Microsoft, who

reflected about what made their workdays good and typical, and self-reported about how they spent their time on various activities at work. We developed conceptual frameworks to help define and characterize developer workdays from two new perspectives: good and typical. Our analysis confirms some findings in previous work, including the fact that developers actually spend little time on development and developers' aversion for meetings and interruptions. It also discovered new findings, such as that only 1.7% of survey responses mentioned emails as a reason for a bad workday, and that meetings and interruptions are only unproductive during development phases; during phases of planning, specification and release, they are common and constructive. One key finding is the importance of agency, developers' control over their workday and whether it goes as planned or is disrupted by external factors. We present actionable recommendations for researchers and managers to prioritize process and tool improvements that make good workdays typical. For instance, in light of our finding on the importance of agency, we recommend that, where possible, managers empower developers to choose their tools and tasks.

4.1 Introduction

Satisfied developers are more productive and write better code [Amabile and Kramer, 2011; Graziotin et al., 2014a,b, 2015a]. Good workdays increase developer job satisfaction [Sheldon et al., 1996]. Understanding what differentiates good workdays from other days, especially atypical days, will help us make good days typical. This work seeks just this understanding. Understanding typical and atypical workdays will enable us to establish a baseline for comparison with other developer workdays and make more informed decisions about process improvements.

Development is a multistage process with complicated interactions across the stages. These interactions mean that we cannot consider each stage in isolation, but need consider the process as a whole. We need a holistic understanding of how software developers spend their time at work. Without a holistic understanding, one might think that developers, because they "develop", spend most of their time writing code. However, developers spend surprisingly little time with coding,

9% to 61% depending on the study [Astromskis et al., 2017; Gonçalves et al., 2011; Meyer et al., 2017a, 2014; Perry et al., 1994b; Singer et al., 2010; Xia et al., 2017]. Instead, they spend most of their time collecting the information they need to fulfill development tasks through meetings, reading documentation or web searches, helping co-workers, and fulfilling administrative duties. The conventional wisdom is that email is a big source of distraction and frustration. We show that, to the contrary, email activity has little effect on a workday's perceived goodness (Section 4.5.1). Hence, focusing just on one development activity can miss important opportunities for productivity improvements.

We have therefore set out to better understand how to make good days typical to increase developer job satisfaction and productivity. Since a review of existing research revealed no work that attempted to define or quantify what a good and typical developer workday is, we studied developers' workdays from these two new perspectives [1]. We conducted a large-scale survey at Microsoft and asked professional software developers whether they consider their previous workday to be good and typical, and related their answers and reflections to their self-reports of the time spent on different activities at work. From now on, when we describe good and typical developer workdays, we refer to developers' self-reports; we discuss the validity of this method in Section 4.4.3.

We received 5971 responses from professional software developers across a four month period. From these responses, we developed two conceptual frameworks to characterize developers' good and typical workdays. When we quantitatively analyzed the collected data, we found that two main activities compete for developers' attention and time at work: their main coding tasks and collaborative activities. On workdays that developers consider good (60.6%) and typical (64.2%), they manage to find a balance between these two activities. This highlights the importance of agency, one of our key findings that describes developers' ability to control their workdays, and how much they are randomized by external factors such as unplanned bugs, inefficient meetings, infrastructure issues.

[1] We *intentionally* do not list our own definitions of good and typical workdays since one aim of this work is to understand the characteristics of these workdays, and how developers assess and define them.

Our work provides researchers and practitioners with a holistic perspective on factors that influence developers' workdays, job satisfaction and productivity. In the paper, we discuss five main recommendations for managers to make good workdays typical. Overall, it is important to remove and reduce obstacles that block developers from creating value and making progress. Our findings confirm and extend recent related work (*e.g.*, [Graziotin et al., 2017; Mark et al., 2008b; Meyer et al., 2014]), including that the most important impediments that require attention are inefficient meetings, constant interruptions, unstable and slow systems and tools, and administrative workloads. Conversely, some factors believed anecdotally to be a problem, such as email, in fact have little effect on how good or typical a workday is perceived to be. Since we found evidence that meetings and interruptions are not bad overall as their impact depends on the project phase, we conclude that they do not have to be minimized at all times. For instance, we can better support the scheduling of meetings and help find more optimal slots depending on the project phase or current workday type. Also, improving developers' perceptions of the importance and value of collaborative work can reduce their aversion against activities that take time away from coding. For example, managers can include developers' contributions to other teams or (open-source) projects when they evaluate them in performance reviews. Finally, giving developers enough control over how they manage their work time is important to foster job satisfaction at work. This can, for instance, be achieved by allowing flexibility in selecting appropriate work hours, locations of work, and tasks to work on.

The main contributions of this paper are:

- Two **conceptual frameworks** that characterize developers' workdays from two new perspectives: what makes developers consider workdays good and typical.

- Results from **5971 self-reports** from professional software developers about how they spend their time at work. The number of responses is an order of magnitude bigger than previous work and allows us to replicate results from previous work at scale, and to uncover nuances and misconceptions in developers' work.

- **Quantitative evidence** identifying factors that impact good and typical workdays for software developers and the relationships between these factors, workday types, and time per activity.

- **Recommendations** that help researchers and practitioners to prioritize process and tool improvements that make good workdays typical.

4.2 Research Questions

Our research is guided by the following main research question: **What is a good and typical workday for developers?** We formulated subquestions to approach the main research question from different perspectives. First, we want to find out qualitatively what factors impact what developers consider as good and typical in a workday:

RQ1: What factors influence good and typical developer workdays and how do they interrelate?

While much related work has looked into how much time developers spend on various work activities (Section 4.3), we want to investigate how developers spend their time differently on days they consider good and typical:

RQ2: How do developers spend their time on a good and typical workday?

The large dataset of 5971 survey responses allows us to compare the time a developer spends on different activities with other developers. We want to group developers with similar workdays together and use other responses from the survey to describe and characterize these groups as workday types:

RQ3: What are the different types of workdays and which ones are more often good and typical?

As described in the related work section, developers spend a lot of time at work in development unrelated activities, such as meetings and interruptions. We want to further investigate the impact of these collaborative aspects on good and typical workdays:

RQ4: How does collaboration impact good and typical workdays?

4.3 Related Work

Related work can broadly be classified into research on developers' workdays and factors that impact these workdays. To avoid repetitions in this thesis, we present and summarize the related work in Section 1.6 of the synopsis. Specific to this publication is previous work on how job satisfaction and happiness influence developers' work, which is why it is presented below.

4.3.1 Influence of Job Satisfaction on Developers' Workdays

What is often left out from research about factors influencing workdays are human aspects, such as developers' job satisfaction and happiness. Job satisfaction is a developer's attitude towards the *general* fulfillment of his/her expectations, wishes and needs from the work that he/she is performing. One important factor that influences job satisfaction is the sum of *good and bad* workdays, which we define as the degree to which a developer is happy about his/her *immediate* work situation on the granularity of a single day. The developer's affective states, such as subjective well-being, feelings, emotions and mood, all impact the assessment of a good or bad workday. Positive affective states are proxies of happiness and were previously shown to have a positive effect on developers' problem solving skills and productivity [Amabile and Kramer, 2011; Graziotin et al., 2014a,b; Müller and Fritz, 2015]. Similarly, aspects of the job that motivate developers or tasks that bring them enjoyment were also shown to lead to higher job satisfaction and productivity [Beecham et al., 2008; Kim and Choe, 2019]. Self-reported satisfaction levels of knowledge workers [Mark et al., 2016b], and more specifically, self-reported affective states of software developers [Graziotin et al., 2015a], have further been shown to be strongly correlated with productivity and efficiency. Similarly, developers' moods have been shown to influence developers' performance on performing programming tasks, such as debugging [Khan et al., 2011]. However, it is unclear how these and other factors influencing developers' workdays affect their assessment of whether a workday is good or bad. Ideally, we would use this knowledge to increase the number of good, positive workdays and reduce the negative ones.

Previous psychological research connected positive emotions with good workdays [Amabile and Kramer, 2011] and satisfaction [Fisher, 2000]. When studying the relationship between positive emotions and well-being, hope was found to be a mediator [Fredrickson et al., 2008; Ouweneel et al., 2012]. Positive emotions at work were further shown to increase workers' openness to new experiences [Kahn and Isen, 1993], to broaden their attention and thinking [Fredrickson, 1998; Ouweneel et al., 2012], and to increase their level of vigor and dedication [Ouweneel et al., 2012], yielding higher work engagement and better outcomes. Sheldon et al. [1996] have further shown that on good days, students feel they have higher levels of autonomy and competence, which also results in better outcomes. One goal of the reported study is to learn how developers assess good workdays and what factors influence their assessment. Amongst other results, we found that on good workdays, developers succeed at balancing development and collaborative work, and feel having spent their time efficiently and worked on something of value (Section 4.5.1).

There is also research indicating that good and typical workdays are related. For example, knowledge workers were shown to be more satisfied when performing routine work [Mark et al., 2014; Melamed et al., 1995]. Contrarily, a literature review on what motivates developers at work, conducted by Beecham et al. [2008], found that the variety of work (differences in skills needed and tasks to work on) are an important source of motivation at work. Similarly, recent work by Graziotin et al. [2017] found that one of the main sources of unhappiness are repetitive and mundane tasks. In this paper, we also investigate the factors that make developers perceive their workdays as typical (Section 4.5.2), and explore the relationship between good and typical workdays (Section 4.6).

4.4 Study Design

To answer our research questions, we studied professional software developers at Microsoft. Microsoft employs over thirty thousand developers around the globe with more than a dozen development centers worldwide. The teams follow a broad variety of software development processes, develop software for several

platforms, develop both applications and services, and target private consumers and enterprise customers.

4.4.1 Survey Development Using Preliminary Interviews

To study developer workdays in a subsequent survey, we needed a taxonomy of activities they pursue. We started with the taxonomy of activities by LaToza et al. [2006] in their study of developer work habits. To validate and potentially enrich this taxonomy, we contacted a random sample of developers at various levels of seniority across many teams and scheduled half an hour to interview them about their activities at work, conducting ten interviews in total. In each interview, we first asked the developer to self-report and describe all of the activities that they engaged in during the previous workday, including the type of activity, the reasons for the activity, the time spent in the activity, and what time of day the activity occurred. We encouraged them to use email, calendars, diaries etc. as these act as "cues" [Tourangeau et al., 2000] and have been shown to reduce interview and survey measurement error [Bellezza and Hartwell, 1981; Bradburn, 2010; Hudson and Davis, 1972; Schwarz and Oyserman, 2001; Tulving and Pearlstone, 1966]. We then asked interview participants to list additional activities that they engage in, regardless of frequency or duration.

After gaining the approval of Microsoft's internal privacy and ethics board, we conducted interviews with developers until the *data saturation point* was reached [Babbie, 2015]. That is, once new interviews yield no additional information, further interviews will yield only marginal (if any) value [Guest et al., 2006]. The set of activities saturated after seven interviews, but we conducted ten to increase our confidence that we had captured all relevant activities. Once we had collected all of the activities, two of the authors grouped them into activity categories using a card sorting approach [Spencer, 2009].

4.4.2 Final Survey Design and Participants

To increase our understanding of developers' workdays and what makes them good and typical, we broadly deployed a survey to developers at Microsoft. We followed guidelines by Kitchenham and Pfleeger [2008] for surveys in software

engineering and based the questions on our insights from the interviews. Our survey comprised four main sections: (1) We first asked about *demographics*, including team, seniority, and development experience. (2) Next we presented respondents a list of activities (those we developed in the interviews) and asked them to indicate how much *time they spent in each activity on their previous workday.* We allowed respondents to write in additional activities if they had an activity that was not covered by our taxonomy. (3) Third, we asked if the previous workday was a typical day or not and if they considered it to be a good day. In both cases, we asked them to explain why as an open response. (4) Finally, we asked a number of additional questions about their day, including how many times they were interrupted, and how many impromptu meetings occurred. In an effort to minimize the time required to complete the survey and avoid participant fatigue, only a random subset of the questions in the fourth category were shown to each respondent. In total, each question in the fourth category was answered by a random 10% subset of respondents. Our goal for the survey was to take only five to ten minutes to complete. After the study was completed, the online survey tool indicated that the median time to complete the survey was just over seven minutes.

First, Microsoft's ethics and privacy board reviewed our survey. To pilot the survey and identify any potential problems, we then sent the survey to 800 developers over the course of one week with an additional question asking if any aspect of the survey was difficult or confusing and soliciting general feedback. After examining the responses, we made small wording changes for clarity and also confirmed that our activity list was complete. Since the changes were very minor, we also included the pilot responses in our analysis. In an effort to make our study replicable, we provide the full survey in the supplementary material [2].

We then sent out 37,792 invitations to complete the survey by sending approximately 500 invitations on a daily basis over the course of roughly 4 months. Developers were selected randomly *with replacement*, meaning that it was possible that a developer would receive the survey multiple times over the course of the study (though never more than once on a given day). Each

[2]Supplementary material: `https://doi.org/10.5281/zenodo.1319812`

developer received a personalized invitation via email that explained who we were and the purpose of the survey. To encourage honest responses and improve participation rate, survey responses were anonymous. In the invitation email and survey description, we explicitly stated that participation is voluntary, the survey is completely anonymous, all questions are optional, and that only aggregated and no individual data will be shared with collaborators at Microsoft. Participants could also contact us in case they had any questions or concerns. Even though the survey was anonymous, 43.8% of respondents choose to reveal their identity. Among them, only 6.6% responded twice and none repeated more than once. In Section 4.8, we discuss potential threats of this study design choice. We analyzed the responses in the unit of a workday, not a developer.

We used a one sample continuous outcome confidence interval power analysis to determine our required sample size [Daniel and Cross, 2018]. To achieve a confidence level of 95% for a 5% confidence interval, the power analysis indicated that we needed 385 responses. Since we were not sure ahead of time the exact ways that we would be partitioning and comparing the responses, we aimed for ten times that amount. In total, we sent 37,792 survey invitations and received 5,971 responses. This is a response rate of 15.5%, which is in line with response rates reported by other surveys in software engineering literature [Punter et al., 2003]. From the 5,971 responses we collected in the survey, 59.1% of the developers stated they are junior and 40.5% senior developers. 0.4% or 26 did not specify their seniority level. Respondents reported an average of 10.0 years ($\pm$7.48, ranging from 0.5 to 44) of experience working in software development.

4.4.3 The Validity of Self-Reported Data

Collecting time-use data can be achieved through various methods, including observations, automated tracking, and self-reporting. We decided to ask developers for self-reports, for the following reasons: self-reports (1) scale better than observations to have a representative sample, (2) they collect a more holistic view compared to using time tracking software that misses time away from the computer (which was shown to be on average about half of a workday for developers [Meyer et al., 2017a]), and (3) since we investigate developers' individual perceptions of good and typical workdays, it makes sense to compare those perceptions with their own estimations of how they spend time. Further, self-reported data is also common in large-scale time-use surveys, such as the American Time Use Survey [Stinson, 1999]. However, self-reports on behaviors and time spent are profoundly influenced by the question wording, format and context, and can, thus, be unreliable [Schwarz and Oyserman, 2001]. To overcome these risks, we carefully designed the self-report questions based on recommendations from related work, especially Menon [1994]; Schwarz and Oyserman [2001], and we test-run our questions first with ten interviewees before running the actual survey study.

We *intentionally* asked respondents to self-report about the activities of the previous workday instead of asking more generally. This was a *conscious* methodological design decision based on the following reasons. First, the previous day is recent, thereby increasing recollection accuracy. This holds true even if the self-report is about the Friday the week before in case respondents answer on a Monday. According to Tourangeau et al. [2000], by far the best-attested fact about autobiographical memory is that the longer the interval between the time of the event and the time of the interview or survey, the less likely that a person will remember it. Second, a day is a short period of time to recall, and a large body of research on surveying and recollection has found that when the reference period is long, respondents tend to use heuristics and estimation of frequencies rather than concrete occurrences [Blair and Burton, 1987; Bradburn et al., 1987; Menon, 1994; Schwarz and Oyserman, 2001]. This can decrease validity, as Menon [1994] found that "to the extent that behavioral frequencies are reported

based on inferential heuristics, they are *judgements* and are subjective". Being asked how many times one went out to eat last week, most people will likely count concrete instances, whereas if the reference period is last year, they will almost certainly estimate based on heuristics. Lastly, even if a respondent does recount concrete events, larger reference periods can fall prey to a phenomenon known as "telescoping" whereby a notable event is remembered as occurring more recently than it actually did [Neter and Waksberg, 1964; Sudman and Bradburn, 1973]. By using the period of a single day, events are less likely to cross a "night boundary" and be attributed to the wrong day [Bradburn et al., 1987].

We encouraged participants in the interviews and survey to use their email clients, calendars, task lists, diaries etc. as "cues" [Tourangeau et al., 2000] to improve their recall of their previous workday and reduce measurement errors [Bellezza and Hartwell, 1981; Bradburn, 2010; Hudson and Davis, 1972; Schwarz and Oyserman, 2001; Tulving and Pearlstone, 1966]. Finally, we asked respondents to self-report the times spent in minutes rather than hours so that they were forced to recall the events in more detail, as the unit of time in response has shown to have an impact on recollection accuracy [LeBoeuf and Shafir, 2009; Schwarz and Oyserman, 2001].

4.5 Conceptual Frameworks

In this section, we answer **RQ1** and present the results from investigating survey respondents' self-reports of what made their previous workday good and typical. We organized the factors influencing developers' workdays as conceptual frameworks and describe them using representative quotes and examples.

4.5.1 Developers' Good Workdays

To identify factors that influence what a good workday is to developers, how they relate to each other, and how important each factor is, we asked survey respondents the following question: "Would you consider yesterday a good day? Why or why not?".

Data Analysis

We coded the responses to the question to a binary rating of either good or not good. Due to the formulation of the question, not good workdays could either refer to an average or a bad workday. From now on, we describe not good workdays as *bad* for better readibility. 5013 participants answered the question; 60.6% (N=3039) stated their previous workday was good and 39.4% (N=1974) stated it was bad.

We qualitatively analyzed the cleaned responses from participants who provided an explanation for what made their workdays good or bad (21.1% did not provide an explanation). We developed a **coding strategy**, applying Open Coding, Axial Coding, and Selective Coding as defined by Corbin and Strauss' Grounded Theory, as follows [Strauss and Corbin, 1998] [3]. The first author Open Coded the entire set of 4005 responses on what made participants' previous workday good or bad, using a quote-by-quote strategy where multiple categories could be assigned to each quote. Responses that could not distinctively be mapped to a category were discussed with the other authors. Before starting the first Axial and Selective Coding iteration, the authors familiarized themselves with the categories that resulted from the Open Coding step, by looking at 10-30 representative responses (*i.e.*, quotes) per category and the number of responses that the first author Open Coded to each category. We then discussed the relationships between these categories in the team (usually with three or all four authors present). This included drawing out the factors and their relationships on a whiteboard, which we collected as memos. During that process, we heavily relied on the quotes and regularly consulted them for additional context and details about the identified relationships. The process was iterative, meaning that whenever the Axial and Selective Coding steps resulted in updates to the Open Coding categories, the first author re-coded participants' responses, and we did another iteration of Axial and Selective coding. After five iterations, we used the memos, factors that resulted from the Axial Coding and high-level factors

[3]Since we applied all components of Straussian's Grounded Theory approach in our analysis but the outcome of this analysis was a conceptual framework instead of a theory, the most accurate description of our analysis is that we used Grounded Theory as a "methodological rationale" [Charmaz, 2014] or "à la carte" [Stol et al., 2016].

(that resulted from the Selective Coding) to create a conceptual framework to characterize developers' good workdays.

Conceptual Framework

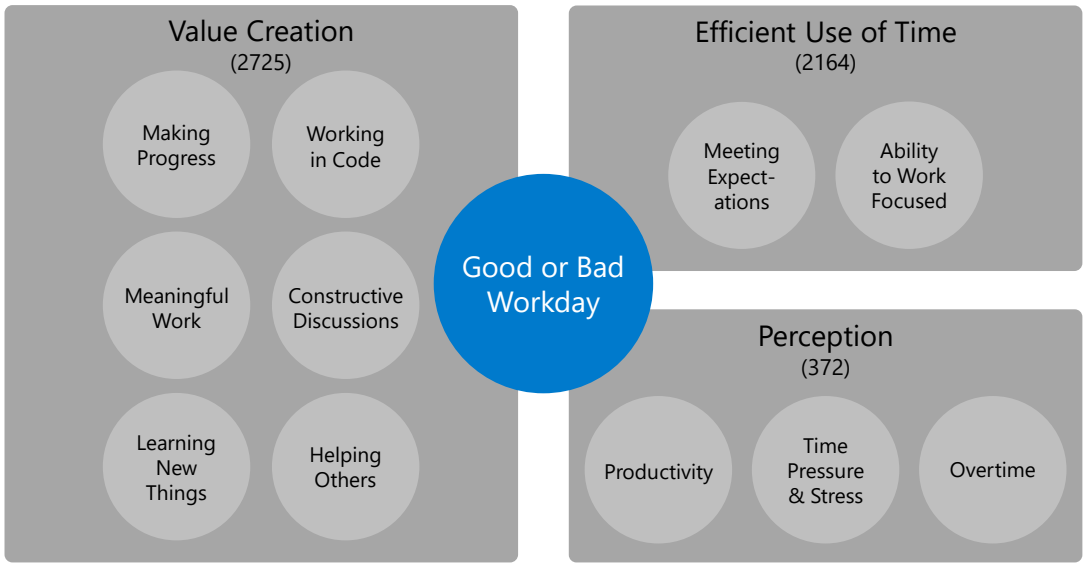
From applying our coding strategy, we identified 11 factors impacting developers' assessment of a good workday. We organized these factors into three high-level factors, (1) *value creation*, (2) *efficient use of time*, and (3) *sentiment*. The first two high-level factors were fairly obvious since respondents usually described good workdays when they considered their work as meaningful and/or did not waste their time on meaningless activities. A few important factors, however, did not fit into these two high-level factors. They are related to respondents' personal feelings and perceptions of their overall work, which we grouped as the third high-level factor. Initially, we thought that *quality* is another important factor, since some respondents described good workdays as days they improved the quality of the software or did not break something. However, we realized that these statements on quality were very rare (0.3% of responses) and that respondents described them as one form of working on something of value.

In Figure 4.1, we visualize the conceptual framework for good workdays. Each of the 11 factors (light gray) influences one of the three high-level factors (dark gray), and they in turn influence whether developers perceive a workday as good. The numbers in parentheses are counts for the number of responses that we categorized into each high-level factor (total N=4005). Since the identified factors are based on responses to an open question, the reported numbers and percentages in this section should only serve to give a feeling about how prevalent each factor is in respondents' assessment of good workdays, rather than exact measures (reality might be higher).

Now, we provide representative examples and quotes to describe the factors and explain how we derived the conceptual framework based on survey responses.

Value Creation To decide whether their workday was good, respondents most often evaluated if they were effective and if they created something of value (68.0%, N=2725 of the 4005 responses to the question). Creating value, however,

Figure 4.1: Conceptual framework for good workdays. The 3 high-level factors are visualized as square layers; outer layers influence the inner layers.



means different things to developers. In 35.6% (N=1425) of the responses, developers considered their workday good when they managed to produce some form of outcome or accomplishment. Participants typically described a good workday being one when they make a lot of **progress** on or complete their main tasks (similar to previous work [Amabile and Kramer, 2011]):

> "*Made some good progress, [the] project is coming together, checked in some tests.*"
> - S105 [good]

Many responses (13.8%, N=553) mentioned that developers feel good about their workday when they can spend parts of it **working in code**, rather than on other activities, such as meetings, interruptions, or administrative tasks. For 6.4% of all responses (N=257), creating value was described as working on something developers deem important and **meaningful** enough to spend their time on. This could be tasks that let the project progress, process improvements that make the team more efficient, improving the quality of the product, or a feature that they consider valuable for end users:

> "*I was able to help influence a decision that I thought was important.*" - S1658 [good]

While meetings were often not considered a good use of their time (as discussed in more detail in Section 4.5.1), 189 responses (4.7%) described a good workday to be when developers participated in good, **constructive discussions**, when important decisions on the project were made, or when connections could be made that are valuable in the future:

> "*Meetings were productive, and we made some new connections with partners that seem promising. I also had a good chat with a former manager/mentor.*" - S483 [good]

Workdays where developers **learned something** new or that increased their understanding of the project or codebase were also considered good, as 4.7% (N=188) of the responses described learning a valuable investment into the developers' or project's future. Similarly, days when developers could **help a co-worker** to learn something new or unblock someone from a problem they are stuck with was generally considered positive and rewarding (4.7%, N=188). However, spending too much time helping others reduces the time they can spend on making progress on their own tasks:

> "*I spent too much time helping team members and not enough on my scheduled tasks.*" - S1880 [bad]

Efficient Use of Time A second high-level factor for considering what a good workday is, is how efficiently developers manage to spend their time (54.0%, N=2164). A developer's workday can be organized in various ways, and there are numerous external and personal aspects that compete with each other for the developer's attention and time. This impacts whether a **workday goes as expected** and influences the developer's **ability to focus** on the main coding tasks. Respondents mentioned that changes to their planned work or deviations from their usual workday are often negatively perceived. Especially, unexpected, urgent issues in a deployed system puts pressure on developers to resolve them quickly (11.6%, N=464):

> "*Started off with a live-site issue (still unresolved), then went to a [different] live-site issue (still unresolved), then I actually got a few minutes to work on the [main] task.*" - S3158 [bad]

Interruptions from co-workers and distractions such as background noise in open-plan offices were described in 13.8% (N=552) of the responses to negatively influence developers' ability to focus or work as planned:

> "*Too many interruptions/context switches. I need a continuous block of time to be really productive as a coder, but I find I get distracted/interrupted more than I'd like.*" - S1066 [bad]

Similarly, long meetings or meetings spreading over the whole day, with very little time in-between to work focused, were another regularly mentioned reason (12.2%, N=491).

10.3% (N=411) of the remaining responses mentioned further reasons for bad workdays that were not numerous enough to be coded into a new category. This includes time lost due to infrastructure issues, outdated documentation, spending too much time to figure out how something works and being blocked or waiting for others (similar to [Graziotin et al., 2017]). Unlike what one might expect from previous work [Barley et al., 2011; Dabbish and Kraut, 2006; Mazmanian, 2013], emails were surprisingly only rarely mentioned as a reason for not being a good workday (1.7%, N=69).

Perception 9.3% (N=372) of all responses about good workdays were related to developers' positive or negative perceptions of their overall work; their productivity, time pressure, and working hours. For example, in 4.7% (N=187) of the responses developers mentioned that they felt **productive** or unproductive on a particular workday, and not specifying what factors contributed to their feeling.

> "*Yes, I was productive and felt good about what I'd done.*" - S322 [good]

In 102 (2.5%) responses, developers considered workdays to be better when they had a good **balance of handling stress**. This includes not trying to meet a tight deadline and not having a too high **time pressure**:

> "*Considering we are not [on] a tight deadline, working in a relaxed fashion and coding were quite enjoyable.*" - S1654 [good]

Time pressure was recently also shown as a major cause of unhappiness of developers [Graziotin et al., 2017]. 2.2% (N=87) of the responses described workdays requiring to **work overtime** as bad. Reasons for what causes overtime work are tight deadlines and having full task lists.

In Section 4.7, we make recommendations about how to leverage these results to make good workdays typical.

4.5.2 Developers' Typical Workdays

To learn how different factors impact developers' assessment of typical workdays, we asked survey respondents the following question: "Was yesterday typical?". To answer the question, they could either select *yes* or *no*. Respondents who picked *no* were asked to provide a reason for their assessment in a free-form textbox.

Data Analysis

5876 participants responded to the question; 64.2% (N=3770) considered their workday as typical and 35.8% (N=2106) as atypical. 2008 of these 2106 participants (95.3%) then explained what made their workdays atypical. To qualitatively analyze these factors and their relationships, we used the **same coding strategy** that we described in Section 4.5.1 to characterize good workdays.

Conceptual Framework

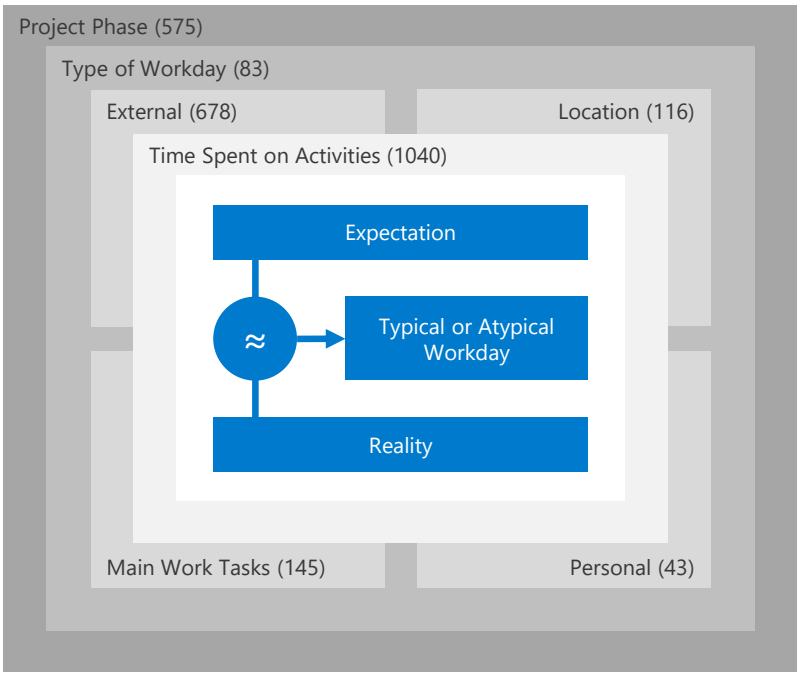
Here, we describe how we developed the conceptual framework that characterizes typical workdays. As respondents were very vocal about meetings and interruptions to describe main reasons for atypical workdays, we initially thought that a key component for the assessment is whether the developer has control over the factor. However, from our discussions and after coding all responses, we realized that the key component is the match between a developer's **expectation** of how a workday will be and how it actually was in **reality**. Externally influenced

factors are just one factor to influence this match. If the mismatch is large, developers consider the workday atypical. Also, we initially thought that the resulting 7 high-level factors that we identified through our Axial and Selective Coding all directly influence the assessment of typical workdays. However, we noticed that they also influence each other: The current project phase impacts the distribution over the different workday types. The workday type and subsequent factors (external, tasks, location and personal) in turn influence the developer's expectation of how much time is spent in activities. We noticed that it was usually not the activity itself that impacted the assessment, but whether the developer spent more, less or about the same amount of time on it than usual and what the developer expects. The relationships between the layers (*i.e.*, factors) were discovered through extensive discussions in the whole team during the Axial Coding steps, where we discussed the categories that resulted from the Open Coding process in relation with the representative quotes from each category.

In Figure 4.2 we visualize the conceptual framework, including the 7 high-level factors as gray layers. The different gray shades show how each layer influences the inner layers above. The counts in parentheses denote the number of participants whose response we coded into the factor (total N=2008). We explain the conceptual framework, by providing representative examples and quotes to describe the factors.

Project Phase In 28.6% (N=575) of the responses, developers assessed their previous workday as atypical, providing the current project phase as the reason. In agile software development, an iteration usually starts with a planning and specification phase, is followed by a development or quality/stabilization phase, and then finalized with the release phase. Respondents, however, often perceived their workdays as atypical when the phase was not development (22.3%, N=448). Since non-development phases occur less frequently, are usually shorter, and often let developers code less than in development phases, they are perceived atypical:

Figure 4.2: Conceptual framework characterizing typical workdays. The main factors are visualized as layers; the outer layers influence all inner layers.



> "*We are in the planning phase now and each day is different: There is a lot more focus on evaluations, specs, meetings during this phase. This would significantly differ from our workday during coding milestones.*" - S2243

These non-development phases are perceived as "slower", which is why developers spend more time on activities that usually fall short during development phases, such as training, evaluating frameworks, writing documentation, or working on low-priority tasks.

Since developers often describe meetings and interruptions as unproductive, prior work concluded that they are bad overall [Bailey et al., 2001; Czerwinski et al., 2004; González and Mark, 2004; Mark et al., 2008b; Meyer et al., 2014; Parnin and Rugaber, 2011]. We are refining these earlier findings with our results that suggest the impact of meetings and interruptions on productivity depends on the project phase. Respondents described that during phases of specification, planning and releases, they are common, but constructive:

> "*In planning stage, not development stage. Spent way more time in meetings than normal, but they were productive meetings.*" - S1762

As some teams at the studied organization do not employ separate teams dedicated to operations, developers also have to take part in a servicing phase, usually for about one week every couple of weeks. During that week, they are on-call for urgent and unexpected issues, which respondents often also regarded as atypical:

> "*I am currently on-call as [an] incident manager. It was typical for on-call rotation, but that happens only once every 12 weeks.*" - S2447

While many respondents described the current phase to be atypical, few mentioned that the amount of time they spent on certain activities was unusual for the phase they are currently in. For example, spending an unusual high amount of time with coding during a planning phase felt atypical.

Type of Workday The workday type is another factor that influences whether a developer considers a workday as typical. While this factor is not as prominent as the project phase that influences it, the workday type was emphasized in 4.1% (N=83) of the responses. Certain teams or individuals organize their work into days where they mainly focus on a single activity, such as coding, bug-fixing, planning or training. During these days, developers avoid pursuing activities that distract them from their main activity, such as emails or meetings. Specific days where the team would not schedule any meetings were mentioned most prominently:

> "*We're not allowed to schedule meetings on Thursdays.*" - S419

> "*On Mondays I have a late night coding night.*" - S159

These well-appreciated "no-meeting days" were often scheduled during development phases to allow developers to focus primarily on making progress on their coding tasks.

External In 33.8% (N=678) of the responses, developers mentioned external factors they have little control over to contribute to an atypical workday, as they often divert developers from making progress on their main tasks towards other

responsibilities. Among these, **meetings** were most often mentioned (20.5%, N=411), especially when they are unplanned, very long or concerning an unusual topic:

> "*I had literally no meetings and spent the entire day refactoring legacy code. That is unusual.*" - S2394

Lack of knowledge, *e.g.*, when having to work on a task in an unfamiliar codebase or with a new framework, is another factor that can make workdays atypical (3.9%, N=78), as developers have to stop working on their tasks and spend time **learning or attending trainings** about a topic:

> "*I was working in a very unfamiliar system; so I spent a lot of time struggling with how to do basic things like building. Normally, I work in a system that I'm familiar with.*" - S1490

Fixing **infrastructure** problems or performing maintenance (*e.g.*, internet issues, installing updates) were described as the reason for an atypical workday by 3.9% (N=78) of the responses:

> "*Hardware maintenance/setup is not part of my typical responsibilities. I don't think that I am typically randomized to this extent.*" - S1306

Similar to our finding on what makes a good workday, emails were only rarely mentioned as a reason for an atypical workday (1.4%, N=28). Finally, respondents mentioned other factors that make their work more fragmented and divert them away from their main work tasks, including long or topic unrelated interruptions (3.6%, N=72), and being blocked on a task and having to wait for others (0.5%, N=11).

Location As developers typically work at the organization's location on-campus, any other location they work on is often considered atypical (5.8%, N=116):

> "*I worked from home and was able to make much better progress than normal.*" - S4679

When developers are not working in their usual location, they either work from home, at a client's or partner's office, or they are traveling to their team's remote location in case it is distributed. Working from home was uniformly

described as a well-appreciated opportunity to get tasks done with minimal interruptions, as it is one way to regain some control over the external factors that can randomize a workday.

Main Work Tasks 7.2% (N=145) of responses mentioned the task itself impacts whether they consider workdays typical. This is the case for **unusual tasks**, such as preparing for a demo or presentation, very easy tasks, such as routine work, or very challenging tasks:

> "*I am not normally prepping for a presentation.*" - S1554

As most development teams at the studied organization plan work by assigning work items in Scrum meetings, **unplanned and urgent tasks** are another reason for sidetracking developers:

> "*No, there was a high priority issue. Normally I would try to spend a little bit more time coding.*" - S5057

These unplanned tasks impact developers' workload and can make it harder to meet deadlines, which often results in longer than usual workdays:

> "*16 hour work days are insane, a big chunk of time was spent troubleshooting a sideways deployment to [product].*" - S5309

Personal Few of the responses (2.1%, N=43) used personal factors to explain what made a workday atypical. The only personal reasons that were mentioned are health related, *e.g.*, feeling sick or having a doctor's appointment. After identifying the importance of personal factors for the framework about good workdays, we more closely inspected responses describing typical and atyipcal workdays from that perspective. No respondent described personal factors such as mood, sleep quality or the ability to focus as factors that impact typical workdays.

This surprised us, since previous work identified that these more personal factors can impact the workday organization, focus and productivity of knowledge workers [Althoff et al., 2017; Czerwinski et al., 2004; Graziotin et al., 2017, 2014a], and thus, presumably, also their assessment of a typical or atypical workday.

As respondents did reveal personal factors when characterizing good and bad workdays, and as we ensured respondents anonymity (see Section 4.4.2), reasons for the lack of personal details in this question might be that the survey setting did not prompt them explicitly enough to reflect about personal factors or the other identified factors are more prevalent and have a bigger influence on their assessment of a typical workday.

Time Spent on Activities Finally, the high-level factors described above influence how much time developers spend on different activities. For example, during the development phase, developers typically spend more time writing code, days with many unplanned meetings reduce the time spent on main coding tasks, and unplanned urgent tasks could force a developer to work overtime. We categorized all cases where 51.8% (N=1040) of the responses contained descriptions of spending *more* or *less* time than usual on a certain activity and visualized the top 5 in Table 4.1. On atypical workdays, respondents mostly reported spending more than usual time in meetings and debugging or fixing bugs.

4.5.3 Interrelationship Between Good and Typical Days

In our analysis of related work, we found an interconnection between job satisfaction, goodness, routine, typicality and productivity. First, we developed conceptual frameworks to better understand these factors in isolation. Now, we describe our main qualitative observations on the interrelationship between good and typical workdays, and present quantitative results in Section 4.6.

One key finding is the importance of **control**, *i.e.*, developers' ability to control their workday and whether it goes **as planned/expected** or is disrupted by external factors, such as unplanned bugs, inefficient meetings, or infrastructure issues. While our findings replicate some results from previous work, they help to better understand nuances in developers' work and sort out misconceptions. For example, when we looked more closely at why developers are very vocal about **meetings** and **interruptions** being one of the main detriments to productive work, we found that during non-development phases, they are better accepted and more productive. Another insight from studying factors that influence good

and typical workdays was that the time spent on **email** (as opposed to email content) are rarely the reason for bad or atypical workdays. Finally, developers described **personal** factors only very rarely as reasons for their assessment. This might suggest that developers are not very aware of how their private lives, health and mood impact their work, or they chose not disclose these factors since they are too personal.

Table 4.1: Top 5 activities where respondents reported spending more or less than usual time in on atypical workdays. Percentages are based on all 2008 responses to the question.

Activity Category	**More than Usual**	**Less than Usual**
Meetings	10.2% (N=205)	5.8% (N=116)
Debugging/Fixing Bugs	6.5% (N=131)	1.3% (N=27)
Coding	3.9% (N=78)	5.1% (N=102)
Planning/Specification	1.6% (N=33)	0.2% (N=5)
Learning/Tutoring	1.5% (N=30)	0.1% (N=3)

4.6 Quantitative Analysis

We provide a quantitative analysis of the relationship between good and typical workdays, by comparing them with the time spent in activities (RQ2), with workday types (RQ3), and with collaborative activities (RQ4). Each analysis reuses the same binary ratings for good and typical workdays that were used to develop the conceptual frameworks.

Table 4.2: Contingency table for the relationship between good and typical workdays (WD). The left number in a cell indicates the percentage and the right number in a cell the total number of responses.

	Typical WD		**Atypical WD**		**Total**	
Good WD	39.8%	1989	20.8%	1037	60.6%	3026
Bad WD	23.5%	1175	15.9%	796	39.4%	1971
Total	63.3%	3164	36.7%	1833	100%	4997

Table 4.3: Mean and relative time spent on activities on developers' previous workdays (WD). The left number in a cell indicates the average relative time spent (in percent) and the right number in a cell the absolute average time spent (in minutes).

Activity Category	**All** 100% (N=5928)		**Typical WD** 64% (N=3750)		**Atypical WD** 36% (N=2099)		**Good WD** 61% (N=3028)		**Bad WD** 39% (N=1970)	
	pct	min	pct	min	pct	min	pct	min	pct	min
Development-Heavy Activities										
Coding (reading or writing code and tests)	15%	84	17%	92	13%	70	18%	96	11%	66
Bugfixing (debugging or fixing bugs)	14%	74	14%	77	12%	68	14%	75	13%	72
Testing (running tests, performance/smoke testing)	8%	41	8%	44	7%	36	8%	43	7%	38
Specification (working on/with requirements)	4%	20	3%	17	4%	25	4%	20	4%	20
Reviewing code	5%	25	5%	26	4%	23	4%	24	5%	26
Documentation	2%	9	1%	8	2%	10	2%	9	2%	8
Collaboration-Heavy Activities										
Meetings (planned and unplanned)	15%	85	15%	82	17%	90	14%	79	18%	95
Email	10%	53	10%	54	10%	54	9%	52	10%	57
Interruptions (impromptu sync-up meetings)	4%	24	4%	25	4%	22	4%	22	5%	28
Helping (helping, managing or mentoring people)	5%	26	5%	27	5%	25	5%	26	5%	28
Networking (maintaining relationships)	2%	10	2%	9	2%	12	2%	11	2%	10
Other Activities										
Learning (honing skills, continuous learning, trainings)	3%	17	3%	14	4%	22	3%	19	3%	16
Administrative tasks	2%	12	2%	11	3%	14	2%	11	3%	15
Breaks (bio break, lunch break)	8%	44	8%	44	8%	45	8%	44	8%	45
Various (*e.g.*, traveling, planning, infrastructure setup)	3%	21	3%	17	5%	27	3%	19	4%	25
Total	**9.08** hours		**9.12** hours		**9.05** hours		**9.17** hours		**9.15** hours	

4.6.1 Correlation Between Typical and Good Workdays

First, we created a contingency table (see Table 4.2) to investigate the correlation between good and typical workdays. A Fisher's exact test shows strong statistical significance (p=0.00001324, 95% confidence interval). This means that although typical and atypical workdays are both more likely to be considered good than bad, the percentage of typical workdays that were considered good (62.9%, good typical days over all typical days) is higher than the percentage of atypical workdays that were considered good (56.7%, good atypical days over all atypical days) to a statistically significant degree. Similarly, from studying emotions developers express when writing and commenting issue reports, Murgia et al. [2014] found that surprise, which could be more often experienced on atypical workdays, is associated with negative events.

4.6.2 Time Spent on Activities at Work

Previous research on how developers spend their time at work did not consider whether developers think they were good and typical, or whether they were an unusual representation of work [Gonçalves et al., 2011; Meyer et al., 2017a, 2014; Perry et al., 1994b; Singer et al., 2010]. Hence, optimizing processes and tools without this knowledge is risky, since we might draw wrong conclusions and optimize for bad or atypical workdays. For example, from previous studies we could get the impression that reducing the email workload of developers is of very high importance. However, our study showed that while developers spend time with emails, they do not consider them an important factor that makes workdays bad or atypical. Hence, to answer **RQ2** we asked participants to self-report the time they spent on various activities at work and related them to their assessments of good and typical workdays.

Data Analysis. In the survey, respondents filled out a table with the minutes spent in predefined activity categories. They also had the option to add other activity categories in case they were missing. For the quantitative analysis, we only used responses where the total time spent was greater than zero and smaller than 24 hours. We then calculated the mean and relative time spent per activity

category for all respondents, for respondents who reported they had a typical or atypical workday, and respondents who reported they had a good or bad workday.

Results. In Table 4.3, we visualize the mean number of minutes and relative time (in percent) participants reported having spent on each activity on their previous workday. Column 2 lists an average over all participants, while Columns 3 and 4 consider typical and atypical workdays, and Column 5 and 6 consider good and bad workdays. In total, developers spent on average slightly more than 9 hours at work on their previous workday. While this includes an average of 44 minutes non-work time spent at lunch and with bio breaks, the time spent at work is nonetheless higher than the often considered 8 hours for a regular workday (*e.g.*, [Claes et al., 2018]). Since developers at the studied organization can freely organize their work hours, this might be an indication of developers working overtime, an observation that was previously made for German and Finnish workers who had autonomy over their working time and worked overtime [Kandolin et al., 2001; Lott and Chung, 2016]. Overall, the self-reported 9 hours spent at work is in line with our previous work, where we found that developers' work activities span across 8.5 hours on average, identified through computer interaction time tracking [Meyer et al., 2017a].

Activities are grouped into *Development-heavy*, *Collaboration-heavy* and *Other* activities. A few activities could be grouped into multiple groups, *e.g.*, pair programming and code reviewing. Hence, we define a development-heavy activity as an activity usually performed by the developer alone, and a collaboration-heavy as an activity that usually involves multiple people. Activities categorized as *Other* are usually not directly related to development tasks or working with other people.

Most of the time is spent with development-heavy activities, such as reading or writing code (15%, 84 mins), debugging or fixing bugs (14%, 74 mins) and testing (8%, 41 mins). Developers also spent time collaborating with other, including meetings (15%, 85 mins), emails (10%, 53 mins), and helping or mentoring others (5%, 26 mins).

Comparing quantitative self-reports on time spent on activities across good

and typical workdays confirms the previously established qualitative characteristics of good and typical workdays (see Sections 4.5.1 and 4.5.2). Both, on **good and typical workdays, developers spend considerably more time with development** related activities. For example, the time spent with reading and writing code is 22 minutes higher on typical (compared to atypical) workdays and 30 minutes higher on good (compared to bad) workdays. On typical workdays, developers also spend slightly less time in meetings, with planning or working on requirements, and with learning or in trainings. And on good workdays, they spend about half an hour less in collaborative activities, than on bad workdays.

4.6.3 Workday Types

Looking at average relative time spent in activities for all responses results in the impression that good/bad and typical/atypical workdays are very similar overall. However, respondents described that not all workdays look the same, *e.g.*, when they have no-meeting days, and that this type of workday often influences whether they consider a workday as typical or atypical. Since we did not prompt them to discuss workday types, only 4.1% (N=83) of respondents mentioned it. To evaluate similarities and trends in developers' workdays and to answer **RQ3**, we reused our dataset (see Section 4.6.2) to group responses together where respondents reported spending their time at work with similar activities. We then used responses to other questions to characterize these groups as workday types.

Data Analysis. To identify groups of developers with similar workdays, we run a cluster analysis following steps:

1. For the clustering, we used respondents' self-reports of the relative time spent in each activity category. The data cleaning process is the same as described before in Section 4.6.2. To group the respondents, we used the Partitioning Around Medoids (PAM) clustering algorithm by [Kaufman and Rousseeuw, 1987] in the *pamk* implementation from the *fpc* package[4] in R. We varied the number of clusters (k) from one to twenty. The *pamk*

[4] `https://cran.r-project.org/web/packages/fpc`

function is a wrapper that computes a clustering for each k in the specified range and then returns the clustering with the optimum average silhouette. In our case, the optimal number of clusters was $k = 6$.

2. To describe the inferred six clusters, we used responses to other questions from the survey, including developers' assessments of good and typical workdays, their experience (senior versus junior developer (as defined by the organization position) and number of years of development experience) and their office environment (private versus open-plan office).

3. Finally, we used the cluster descriptions to develop workday types.

Results. In Table 4.4, we present the resulting six clusters, the amount of time developers in each cluster spend on different activities, and additional factors to describe the clusters. Clusters 1 to 3 (C1-C3) are development heavy workdays, while clusters 4 and 5 (C4-C5) include more collaborative aspects. In the following, we describe the clusters as workday types and characterize them considering the factors mentioned above. We also name each workday type to make referencing them easier in the paper.

On a **"Testing Day"** (C1), developers overall spend considerably more time with testing compared to the other days. As testing often requires to also debug and fix code, they also spend more time with coding and debugging compared to other not development heavy days (C4-C6). On "Testing Days", developers spend more time learning new things than the other days. The majority of the developers in this cluster (in our sample, 71%) are junior developers, with 66% considering it a typical workday and 63% a good workday respectively. On a **"Bugfixing Day"** (C2) developers spend significantly more time debugging or fixing bugs (almost 3 hours on average). Similar to the "Testing Day", mostly junior developers are experiencing this workday type (69%), and the developers in this cluster generally thought it was fairly typical (65%) and good (60%). A **"Coding Day"** (C3) is a workday where developers spend a lot of their time reading and writing code, on average about 2.3 hours, and is perceived as good by more developers than the other workdays (74%). This workday type has a higher chance to be perceived as typical, with 72% considering their previous coding

day as typical. 65% of the developers in this cluster are juniors and most of the developers in this cluster do not sit in private offices (60%). The **"Collaborating Day"** (C4) entails spending more time on collaborative activities, especially in meetings, with emails and helping others, than on development heavy days (C1-C3). Meetings, however, account only for about half the time than on a 'Meeting Day". This workday was only perceived as good by half the respondents (50%) and is experienced more often by senior (59%) than junior (41%) developers. On a **"Meeting Day"** (C5), developers spend on average more than 4 hours or 46% of their workday in meetings. The majority of developers in this cluster are senior developers (71%). Also, time spent with emails is higher than on workdays with a bigger focus on development (C1-C3). Overall, developers who experience a meeting workday, spend on average just about one and a half hours in development related activities. Only 50% of the developers in this cluster perceived their previous "Meeting Day" as typical. As only 8.5% of the respondents (N=504) belong to this cluster, developers are less likely to experience a meeting day. Finally, on a **"Various Day"** (C6), developers spend more than 4 hours with activities mapped to the category *Various*. This includes setting up the infrastructure, working in support, and on a deployment. With only 1.6% (N=94) respondents belonging into the cluster it is the rarest workday type, the longest workday, and the workday that was most often considered as atypical and bad.

We make the following observations based on the results:

- The number of good workdays in each cluster confirms again that many developers feel more positive on workdays they can work alone and focus most of their time on development related activities.

- What developers consider a good workday also varies with their seniority. Similarly, senior developers experience more atypical days than junior developers.

- Developers experience development heavy workdays (C1-C3) as more typical than the other workdays.

- Senior developers have more days that include collaboration, such as meetings, planning and specification and are less likely to have development-heavy workdays (C1-C3).
- Overall, average workday lengths are very similar, differing only up to half an hour.
- On average, respondents who experienced a development heavy workday (C1-C3) have about 3 years less development experience.

4.6.4 Collaboration

Previous work has described interruptions as one of the biggest impediments to productive work [Bailey et al., 2001; Czerwinski et al., 2004]. Also, meetings have been shown to account for a big chunk of a developer's workday, but their usefulness and value is often questioned [Meyer et al., 2014]. Since most of the time developers spend in development-unrelated activities is of a collaborative nature, we wanted to better understand how the two most frequent collaborative aspects, meetings and interruptions, impact good and typical workdays (**RQ4**). Note that we did not include the project phase in this analysis, as respondents were not specificly asked to report it in the survey.

Data Analysis. To study how collaboration impacts developers assessments of good and typical workdays, we selected 8 aspects related to interruptions and meetings. For each aspect, we asked a random 10% subset of respondents to self-report the time needed, total number, or percentage. After cleaning the data and removing a few outliers (<1% per aspect), we correlated the aspect with respondents' assessments of good and typical workdays. To test whether there is a significant statistical difference between each, good and typical workdays, and the aspect, we use non-parametric Wilcoxon signed-rank tests (95% confidence interval). We corrected all p-values for multiple hypothesis testing, using the Benjamini-Hochberg correction [Benjamini and Hochberg, 1995]. In the results that follow, we describe the relationship as significant if the p-value is less than 0.05. In these cases, we include the p-value of the Wilcoxon test inline.

Table 4.4: The six workday type clusters. Each column corresponds to a cluster and each row either to the time spent in activities or factors considered to describe the clusters. The left number in a cell indicates the average relative time spent and the right number in a cell the average absolute time spent.

	Cluster 1		Cluster 2		Cluster 3		Cluster 4		Cluster 5		Cluster 6	
	Testing Day		**Bugfixing Day**		**Coding Day**		**Collaborating Day**		**Meeting Day**		**Various Day**	
	24% (N=1451)		23% (N=1344)		22% (N=1319)		21% (N=1216)		8% (N=504)		2% (N=94)	
	pct	min	pct	min	pct	min	pct	min	pct	min	pct	min
Time Spent in Activities												
Development-Heavy Activities												
Coding	11%	63	11%	61	37%	137	7%	37	6%	32	6%	35
Bugfixing	11%	59	32%	170	10%	56	6%	31	4%	20	6%	36
Testing	16%	87	7%	36	6%	35	2%	13	2%	11	4%	25
Specification	4%	19	1%	6	2%	10	9%	47	3%	18	1%	4
Reviewing code	5%	25	4%	23	4%	23	6%	33	3%	14	3%	19
Documentation	3%	16	1%	4	1%	6	2%	9	1%	8	1%	3
Collaboration-Heavy Activities												
Meetings	12%	65	9%	50	9%	50	22%	121	46%	248	7%	41
Email	8%	45	10%	52	8%	42	13%	73	12%	66	8%	44
Interruptions	4%	24	4%	21	4%	19	6%	33	3%	19	3%	20
Helping	4%	24	4%	21	4%	20	8%	44	5%	26	2%	14
Networking	3%	14	1%	8	1%	7	2%	12	2%	12	1%	7
Other Activities												
Learning	7%	36	2%	10	2%	13	2%	11	2%	10	2%	9
Administrative tasks	2%	12	2%	10	1%	7	4%	22	2%	11	2%	8
Breaks	8%	44	9%	50	8%	43	8%	44	7%	37	7%	40
Various	2%	19	3%	14	3%	14	3%	20	2%	15	47%	262
Factors Describing the Clusters												
Workday was good	**63%**		**60%**		**74%**		**50%**		48%		**40%**	
Workday was typical	**66%**		**65%**		**72%**		**61%**		**50%**		**40%**	
Works in a private office	46%		47%		40%		60%		57%		75%	
Is a junior developer	71%		69%		65%		41%		29%		56%	
Years of Experience	8.9		8.8		9.4		12.0		13.0		12.0	
Total time spent (in hrs)	9.2		8.9		9.0		9.1		9.0		9.4	

Results. The mean values for each aspect, presented for good, typical and combined workdays, are shown in Table 4.5. We present some key observations:

(4.5a) Overall, developers had on average 4.66 **interruptions** on their past workday. Surprisingly, on typical workdays, the number of interruptions are significantly higher than on atypical workdays ($p < 0.001$). There is also strong evidence (*i.e.*, a significance correlation) that interruptions are lower on good workdays ($p < 0.001$). Developers who perceived their workday as good experienced about one interruption less than developers on a bad workday.

(4.5b) Previous work has found that developers typically take around 15 minutes to **resume after an interruption** [Van Solingen et al., 1998]. In our survey, respondents self-reported it took them on average less than ten minutes. Overall, the time needed to resume an interruption is similar, independent of whether it is perceived as good and typical.

(4.5c) The longest period developers could **work on code without an interruption** was 47.3 minutes, on average. On good ($p < 0.001$) and typical ($p = 0.002$) workdays, developers get significantly longer chunks of uninterrupted coding time. On atypical and bad workdays, it is on average around 40 minutes.

(4.5d) Developers see meetings often as an inefficient use of time. While the results presented above indicate developers are preferring coding over meetings, they still consider the majority of **meetings as useful**, independently of whether it was a good and typical day. The percentage of useful meetings has a significant impact on if a workday is considered good ($p = 0.035$), but not on if it was typical. On atypical workdays, meetings are generally seen as a bit more valuable, probably because they are more irregular and less routine meetings.

(4.5e) **Impromptu sync-up meetings** are short interruptions from co-workers in the developers' office to answer a question. Respondents experienced on average about two impromptu sync-up meetings on their previous workday.

(4.5f) **Unplanned meetings** are more formal and less ad-hoc than impromptu sync-ups, but are not stored as calendar appointments like planned meetings. Developers rarely have unplanned meetings, overall less than one a day. On good workdays, the number of unplanned meetings is significantly lower than on bad days ($p = 0.002$).

Table 4.5: How meetings and interruptions influence good and typical workdays. Significant relationships between the aspect and good or typical workdays are visualized in bold (α<0.05).

Aspect	Typical	Atypical	Good	Bad	Total
a) Number of Interruptions	**4.80**	**4.43**	**4.30**	**5.24**	4.66
b) Minutes needed to resume work after an interruption (in mins)	8.55	8.81	8.19	9.29	8.64
c) Longest period of uninterrupted coding time (in mins)	**50.3**	**42.4**	**52.9**	**39.3**	47.3
d) Percentage of useful meetings	56.6%	61.7%	**61.2%**	**54.8%**	58.5%
e) Number of impromptu sync-ups	2.15	2.30	2.15	2.50	2.20
f) Number of unplanned meetings	0.61	0.63	**0.51**	**0.84**	0.62
g) Total time spent in unplanned meetings (in mins)	21.1	22.1	**16.6**	**28.8**	21.5
h) Percentage of total time spent in meetings	17.7%	22.7%	**18.4%**	**21.4%**	19.5%

(4.5g) When they do happen, unplanned meetings account for slightly more than a quarter hour on good and almost half an hour on bad workdays. On these bad workdays, unplanned meetings take significantly more time ($p = 0.001$).

(4.5h) Of the total time spent at work, developers spent around 20% **in (and preparing) meetings**. On good and typical workdays, the percentage of time spent in meetings is slightly lower (only significant for good workdays, $p = 0.002$). The highest percentage of time spent in meetings is on days developers perceive as good and atypical, suggesting that the unusual meetings (those that happen on atypical days) are often considered as constructive and valuable (see Section 4.5.1).

4.7 Making Good Days Typical

Our findings contribute new knowledge about factors that impact good and typical workdays and how developers' work differs on these days. In this section, we discuss how researchers and managers can apply them to make good days typical and to improve developers' productivity and job satisfaction in the workplace.

4.7.1 Optimizing Developer Workdays

Our results provide insights into factors that impact how developers' good and typical workdays are perceived and how they define them. This allows individuals, team leads and managers to **prioritize and target actions that improve processes and tools**, and ultimately productivity. For example, many developers are unhappy when they work on mundane or repetitive tasks [Graziotin et al., 2017]. On the contrary, time pressure and deadlines were found to be major causes for stress [Mark et al., 2014; Melamed et al., 1995] and unhappiness [Graziotin et al., 2017]. Our results suggest ways to reduce these bad days and **make good days typical** (i.e. more routine). Generally, it is advisable to minimize administrative tasks and infrastructure issues, and reduce interruptions and meetings. Developers gain back time they can spend on their coding tasks and increase their job satisfaction. This is especially important for junior developers, who tend to spend most of their time with development-heavy activities such as coding, testing and bugfixing. During these tasks, they need to be able to focus for long chunks of time, without interruptions. Uninterrupted work might be easier to achieve in smaller offices shared with only a few colleagues, rather than the currently omnipresent open-plan offices. During phases or days that require a lot of collaboration, coordination and many meetings (*e.g.*, planning, specification and release phase), the team could move to an open-plan office. To accommodate these changing needs for office space, teams could switch with another team that is working with reversed phases.

Our results further suggest ways to **make atypical days good**, *e.g.*, by working from home on tasks that require a lot of attention and focus, by scheduling no-meeting days, by planning to do little coding during planning phases, or by using slow times (*e.g.*, project wrap-up) for side-projects. Working from home was previously shown to increase knowledge worker job satisfaction, since it increases flexibility of working at one's own pace, allows planning active working hours at times of higher focus and motivation, and in quieter work environments [Rožman et al., 2017; Ruostela et al., 2015; Van Der Voordt, 2004]. Days on which meetings are grouped together allow teams to collaborate, discuss and plan intensively, and days when no meetings are scheduled allow teams to work more focused,

reducing context-switching and interruptions [Donohue, 2018; Lange, 2008]. Common in these examples is that individuals and teams organize their work to be atypical on purpose, which made these days more positive and productive. In our analysis of workday types (see Section 4.6.3), we found that senior developers experience more atypical workdays than junior developers. This is likely the case because they experience more "collaborating days" and "meeting days", making it more difficult to plan out and control one's workday. Besides the beforementioned approaches on making meetings more efficient, other research focused on approaches that reduce the amount of time senior developers need to spend with randomization and helping out, leaving more time to work on their main coding tasks. This includes expert-finding systems to interrupt fewer people with inquiries (*e.g.*, [Balog et al., 2006; Mockus and Herbsleb, 2002]), more efficiently exchanging knowledge and learnings (*e.g.*, [Fritz and Murphy, 2010; Robillard et al., 2010]), increasing team awareness (*e.g.*, [Biehl et al., 2007; Jakobsen et al., 2009; Sarma et al., 2003]), and more efficiently dealing with emails (*e.g.*, [Mark et al., 2016a; Sappelli et al., 2016]). Finally, spending more quality time on activities most described as good days (*e.g.*, coding and learning) could further make senior developers' often randomized, atypical workdays good.

From our work, it remains unclear how much **consistency over days** developers expect and how often they appreciate non-routine atypical workdays. We could imagine that occasionally breaking the routine will also avoid boredom and will make work more exciting overall. However, the ever-changing project phases and various activities pursued at work might already be sufficient to avoid routine. Having now a better understanding of the factors that influence developers' assessments of good and typical workdays, future work could explore how the company culture, development processes, developers' gender (*e.g.*, GenderMag [Burnett et al., 2016]) and personas (*e.g.*, Developers' Productivity Perception Personas [Meyer et al., 2017c]) influence how **pronounced these factors** are. For example, standardized questionnaires from psychology could be leveraged to include perspectives on developers' personality (*e.g.*, BFI-2 [Soto and John, 2016]), impulsivity (*e.g.*, UPPS [Whiteside and Lynam, 2001]) or ability to self-control (Cognitive Absorption Scale [Agarwal and Karahanna,

2000]). Getting more insights into the distinctness of these factors could help individuals, teams and managers to prioritize improvements at the workplace better.

4.7.2 Agency: Manage Competition for Attention & Time

Our results highlight that developers face a constant competition for attention and time between their **main coding tasks (*i.e.*, individual productivity) and collaborating with their team (*i.e.*, team productivity)**. While they generally understand the importance of team collaboration, not every developer perceived it as an efficient and productive use of their time. Developers' workdays are often disrupted by activities they have little control and autonomy over (*e.g.*, meetings scattered over the day, interruptions from co-workers asking for help). These disruptions fragment developers' workdays, allowing them to only spend little time focused on their coding tasks before switching to the next topic [González and Mark, 2004; Meyer et al., 2017a]. This increases frustration and tensions in the team. Thus, being able to **freely control how to best organize one's workday** and whether it goes as planned/expected or is disrupted by external factors is important. For example, allowing developers to "veto" meeting requests can help avoid scattering meetings across the workday that leave very little time to make progres in-between them. In Sociology, this notion is known as agency, and describes "the capacity of individuals to act independently and to make their own free choices" [Barker, 2003]. The finding of the importance of agency is in line with previous work in psychology, in which increased autonomy in a high-variety task led to increased satisfaction [Dodd and Ganster, 1996]. Similarly, a study with students showed that higher levels of autonomy are related to higher well-being and better outcomes [Sheldon et al., 1996]. Agency is related to the concept of daily job crafting, the self-induced changes that employees make in their job demands and resources to meet and/or optimize their work goals [Tims et al., 2012]. These changes can have positive impacts on motivation [Petrou et al., 2012] and performance [Tims et al., 2012].

Previous work has suggested a plurality of approaches that allow knowledge workers to better manage their own workdays and improve collaboration. This

includes avoiding interruptions from co-workers at in-opportune moments [Züger et al., 2017], reducing distractions from communication channels [Mark et al., 2018; Storey et al., 2017], having flexible work hours [Origo and Pagani, 2008], improving the scheduling of meetings between timezones [Tang et al., 2011; Whittaker and Schwarz, 1999], and making distributed meetings more efficient [Babar et al., 2008; Sutherland et al., 2007]. Since these examples are founded in other knowledge work settings, future work could study how to adapt them to the software engineering domain and allow developers to better control and optimize their time.

Our work refines previous work that found developers consider most meetings and interruptions as unproductive. We found that **during non-development phases, they are common and (usually) productive**. We should actively work on reducing interruptions and meetings during the development phase (and on development-heavy days), and encourage them at times when collaboration is most crucial, such as phases of planning/specification, testing and release. Finally, we learnt that senior developers appreciate collaboration generally more than junior developers, suggesting the **value of collaboration changes with experience**. This finding is in line with recent work that found motivation and job satisfaction at work is changing as individuals age [Rožman et al., 2017].

4.7.3 Evaluation of Contributions at Work

The competing interests between individual productivity and team productivity also highlight potential issues in how we **evaluate success and measure productivity at work**. Our findings indicate that developers who reflect about successful work mostly consider the produced output, and value collaborative work much lower. This is why on good and typical days, the majority of developers reported making a lot of progress on their coding tasks and producing a lot of output (49.4%), rather than focussing on contributing something meaningful (6.4%), learning something new (4.7%), or helping a team-mate to make progress (4.7%). Initial discussions with a few project managers at the corporation confirmed this interpretation. According to them, many teams are starting to consider developers' support to other team-members, and contributions to other

teams and even external (open-source) projects when evaluating the performance in job reviews and salary discussions. This is similar to reports from industry, where developers are not only evaluated by their managers but also their peers, or where developers can recommend a small monetary bonus to another developer for great work done [Goler et al., 2019].

While our study with software developers showed that learning something new was rarely (4.7%) mentioned as a reason for a successful day, a study focusing on Korean knowledge workers found that career satisfaction is predicted by an organization's learning culture and goal orientation [Joo and Park, 2010]. According to Hofstede and Arrindell [1998]'s dimension of masculinity/femininity, masculine cultures focus more on competition and success, while more feminine ones on doing what one likes. These examples suggest that reflecting about (successful) work needs to be considered as a separate variable in future work, to better understand how differences in company culture, job profile, goal orientation and gender could impact how developers reflect about their work.

4.8 Threats to Validity

4.8.1 External Validity

The main threat to the generalizability and validity of our results is the external validity. Since our survey was conducted at only one large organization, it is unclear how representative our results are of the views of all software developers, especially from smaller companies or open-source projects. Microsoft employs over thirty thousand developers, who are working on various projects targeting private and commercial users, and at different stages of their projects. Survey respondents work in teams of different sizes, at various locations, using varying technology and processes. Many teams in the organization have come from acquisitions of smaller companies or startups, and their development processes and tools have remained largely untouched. According to Microsoft's yearly diversity report [5], the distribution of developers in the United States (US) was the

[5] https://www.microsoft.com/en-us/diversity/inside-microsoft/default.aspx

following in 2018: 50.9% Caucasian, 39.2% Asian, 4.5% Hispanic, 2.8% African-American, and 2.6% other. 19.9% identify as female, 80% male, and 0.1% as other. At the time of taking our survey, all participants resided in the greater Seattle area in Washington state, US. Thus, while all of the developers we studied are part of a single organization, we are confident to achieve some level of external validity and generalizability, based on the variation in paricipants' context, environment, culture, and experience, as well as the large sample size. Finally, single-case empirical studies have been shown to contribute to scientific discovery [Flyvbjerg, 2006]. Nonetheless, future work could consider additional variables, such as the company culture and size, and developers' gender, personality, and goal orientation.

4.8.2 Construct Validity

In Section 4.5.1, we described how we analyzed the survey responses using methods common in the Grounded Theory approach by Strauss and Corbin [1998]. One potential threat could be that the Open Coding step was performed by one author only. To reduce bias, we discussed 10-30 representative quotes for each coded category as a team, and collectively mapped all responses that could not distincitvely be mapped to a category. All subsequent steps, including Axial and Selective Coding and the development of the conceptual frameworks, were performed iteratively with three or all four authors present.

4.8.3 Internal Validity

Selecting developers randomly with replacement when sending out the survey invitations via email might pose a risk to internal validity. Since all responses were anonymous, we cannot know the exact number of people who took the survey more than once and how much time was between the two responses. We, however, expect the number to be low, since from the 43.8% of participants who voluntarily revealed their identity, only 6.6% responded twice, and no one repeated more than once. Since the survey invitations were sent out over the course of 4 months, we further also expect the number of responses from the

same participant within subsequent days to be very low. Thus, we believe that the large amount of participants (N=5971) reporting at random workdays and varying stages of their projects is a fairly representative sample of how developer workdays look like. Future work could consider how developers' decision to participate in the survey and more importantly, their answers are affected by their experiences of the previous workday and their personality. For example, previous work showed that well-being on the preceding day can affect the well-being on the subsequent day [Sheldon et al., 1996] and, hence, could have affected our respondents' self-reports of how good their previous workday was. Similarly and with respect to personality, people who are generally more likely to respond to surveys are conscientious, agreeable and open to new experiences [Marcus and Schütz, 2005; Rogelberg et al., 2004; Tuten and Bosnjak, 2001]. Our sample might not accurately represent developers who do not fit into these personality groups.

Another limitation of our study could be that we studied developers' workdays based on their self-reports and only on one day (and in a few cases two days) per developer. We discuss our rationale behind the study design in detail in Section 4.4. We are confident of the soundness of our design, also because our results replicate comparable findings (*e.g.*, time spent on activities and the high cost of interruptions and meetings) from previous work that applied differing methods (*e.g.*, observations, tracking).

The survey structure and formulation might be source of further risks to internal validity. For example, asking developers whether they consider their workday as good and typical after asking them to self-report their time spent at work might have framed participants. Another potential framing threat is the positioning of the demographic questions in the beginning of the survey, which might have caused a *Stereotype Threat* [Steele and Aronson, 1995]. For example, a developer working in an open-plan office and not liking this might have been reminded of the fact when answering the survey. Future work should redress this threat by placing the demographic questions at the end of the survey [Danaher and Crandall, 2008; Stricker and Ward, 2004].

We also acknowledge that typicality and goodness of workdays are not binary

ratings in reality, as several factors could influence the assessment of a workday to be rather typical or rather atypical, for example. However, for the sake of first identifying the factors that influence what contributes to a good and typical workday, we decided that using a dichotomous variable makes more sense. Dichotomization was also described as a standard practice in empirical research, to avoid nominal scale violations [Kitchenham and Pfleeger, 2008]. In the interviews, we noticed that there is no consensus between whether writing code and debugging is the same activity. Hence, in the survey, we asked developers to distinguish between writing code and debugging when reporting the time spent. An additional control question revealed that 58% distinguish between writing code and debugging, while 42% do not.

4.9 Conclusion

We extend previous work on understanding developer workdays by adding two new perspectives: what makes a workday good and typical. We report on aspects that characterize what makes developers consider workdays good and typical, and how managers can leverage them to make good workdays typical. On good workdays, developers make progress and create value on projects they consider meaningful, and spend their time efficiently, with little randomization, administrative work and infrastructure issues. What makes a workday typical is primarily assessed by the match between developers' expectations and the reality. Amongst other aspects, this match is heavily influenced by the time they spend on different activities, external factors they have little control over, and the current development phase. Since developers often complain meetings and interruptions are unproductive, prior work concludes that they are bad overall. Surprisingly, we find that their impact on productivity and job satisfaction depends on the development phase: during specification/planning and release phases, they are common, but constructive. Another key finding is the importance of agency, control over one's workday and whether it goes as planned and expected, or is disrupted by external factors. Our work provides a holistic perspective on how developers think these aspects influence their workdays and helps prioritize

process and tool improvements. For example, one unexpected finding is to de-emphasize email, contrary to what was suggested by related work.

Our results stem from a large-scale survey with 5971 responses, where professional software developers reflected about what made their workdays good and typical, and where they self-reported how they spent their time on the previous workday. In contrast to related work using primarily automated tracking, by using self-reports we capture unanticipated events in each developer's own classification at scale. Our scale also gives us the resolution to uncover nuances, *e.g.*, what makes developers happy and satisfied varies with their seniority.

4.10 Acknowledgements

We thank our study participants for their participation. We also thank the anonymous reviewers and our editor for their valuable feedback.

5

Detecting Developers' Task Switches and Types

André N. Meyer, *Manuela Züger, Chris Satterfield, Katja Kevic, Gail C. Murphy, Thomas Zimmermann, Thomas Fritz*

In submission at FSE 2019,

Contribution: Study design and execution, participant recruitment, tool development, data collection, partial data analysis, and paper writing

Abstract

Developers work on a broad variety of tasks during their workdays and constantly switch between them. While these task switches can be beneficial, they can also incur a high cognitive burden on developers, since they have to continuously remember and rebuild the task context–the artifacts and applications relevant to the task. Researchers have therefore proposed to capture task context more

explicitly and use it to provide better task support, such as task switch reduction or task resumption support. Yet, these approaches generally require the developer to *manually* identify task switches. Automatic approaches for predicting task switches have so far been limited in their accuracy, scope, evaluation, and the time discrepancy between predicted and actual task switches. In our work, we examine the use of *automatically* collected computer interaction data for detecting developers' task switches as well as task types. In two field studies–a 4h observational study and a multi-day study with experience sampling–we collected data from a total of 25 professional developers. Our study results show that we are able to use temporal and semantic features from developers' computer interaction data to detect task switches and types in the field with high accuracy of 87% and 61% respectively, and within a short time window of less than 1.6 minutes on average from the actual task switch. We discuss our findings and their practical value for a wide range of applications in real work settings.

5.1 Introduction

To successfully perform their work, software developers are required to constantly switch between a broad variety of tasks, such as implementing a new feature, answering an email or attending a meeting, with each task requiring its own set of artifacts and applications [González and Mark, 2004; Meyer et al., 2017a; Perry et al., 1994a]. These constant task switches result in a high fragmentation of work, requiring developers to continuously interrupt and later resume their tasks and to relocate the artifacts and applications that are relevant to fulfill the task at hand. Subsequently, developers face a higher cognitive burden, lower performance, and a higher error rate [Bailey et al., 2001; Murphy et al., 2005].

To support developers in their fragmented task work, researchers have proposed approaches that explicitly capture task context–artifacts and applications relevant to the task–and that use this information to then support users by preventing interruptions, easing task resumption, or by recommending relevant artifacts and applications [Bragdon et al., 2010a; Card and Henderson Jr, 1986; Dragunov et al., 2005; Kersten and Murphy, 2006; Sahm and Maalej, 2010; Smith

et al., 2003]. While studies have shown that the explicitly captured task context can lower the cognitive burden on developers and increase productivity [Bragdon et al., 2010b; Kersten and Murphy, 2006], all of these approaches require some form of *manual* interaction of the developer to identify task boundaries, something that developers often forget to do in practice after using such an approach for a few days [Kersten and Murphy, 2006].

To address this issue, few researchers have proposed approaches to *automatically* detect switches between tasks, varying mainly in the features used (*e.g.*, user input or application based), and the method applied (*e.g.*, supervised versus unsupervised) [Mirza et al., 2011a; Shen et al., 2009, 2007]. Yet, the evaluations performed to study these approaches are often fairly limited in terms of the tasks and number of participants, and the results show that it is very challenging to achieve high prediction accuracy of task switches without too many false positives [Mirza et al., 2011a; Nair et al., 2005; Stumpf et al., 2005], or that one has to accept a high deviation in time of 3 to 5 minutes between predicted and actual task switches [Shen et al., 2009, 2007, 2006]. Since these approaches focus on detecting task switches within the IDE only, they are not capturing non-development work, which can account for 39% up to 91% of the time developers spend at work [Astromskis et al., 2017; Gonçalves et al., 2011; Meyer et al., 2017a; Perry et al., 1994a; Singer et al., 2010].

In our research, we extend this work and investigate **(RQ1)** whether we can automatically detect task switches of professional software developers in the field, based on temporal and semantic features as extracted from their computer interaction inside and outside the IDE. We were also interested in classifying the type of task a developer is working on, since the better we understand the context of a task, the better we can support developers. To the best of our knowledge, there has been only one approach so far that looked at the *automatic* classification of developers' activities on a task level [Bao et al., 2018]. Yet, their examination was limited to specific development activities only, and did not consider the whole range of non-development tasks that developers are working on, such as administrative or planning tasks. In our work, we investigate the task types that software developers are working on more holistically, and explore

(RQ2) how accurately we can predict them in the field.

To address our research questions, we performed two field studies: one with 12 professional developers in which we observed their work over a 4-hour period and logged the task switches and types without interrupting their work; and one with 13 professional developers in which we regularly prompted participants to self-report their task switches and types over a period of about 4 workdays and conducted a post-study questionnaire. By varying the study methods, we wanted to achieve a higher generalizability of our results and ensure that we take into account the effects of self-reporting while also capturing the breadth of developers' tasks over multiple days. For both field studies, we also collected the participants' computer interaction using a monitoring tool that we installed on their machine and that was running in the background. From the computer interaction data, we extracted a total of 68 temporal and semantic features. Our analysis of the data shows we can use the automatically logged computer interaction data to train machine learning classifiers and predict task switches with a high accuracy of 87%, and within a short time window of less than 1.6 minutes of the actual task switch. Our analysis further shows that we are able to predict task types with an accuracy of 61%, yet that this accuracy varies a lot by task type. The features based on mouse and keyboard interaction generally hold the highest predictive power, while the lexical features we extracted from the application names and window titles have the least predict power in our approach.

Overall, our work extends previous work with an approach that uses a broader range of features, is evaluated in a field-study with 25 professional developers, and achieves higher accuracy and less delay than previous work. Our results provide evidence for the potential to *automatically* detect software developers' task switches and types in the field. This opens up opportunities for providing developers with task support tools that lower the burden of fragmented work and constant task switching, by reducing task switching and facilitating task resumption, and by greatly complementing existing task support by freeing the developer from the laborious manual task boundary identification.

The primary contributions of this paper are:

- An approach to automatically detect task switches and types based on developers' computer interaction that is not limited to the IDE.
- Two field studies with 25 professional developers demonstrating our approach's potential to detect task switches and types with high accuracy and within a small time window in the field.
- An evaluation of the predictive power of various computer interaction features, including semantic and temporal ones, and a comparison of individually trained models versus a general model.

5.2 Related Work

Work related to our research can broadly be grouped into research that examined the detection of task switches and task types, and into approaches to support task focused work. Based on previous work Bruegge and Dutoit [2004]; González and Mark [2004]; Kersten and Murphy [2006]; Meyer et al. [2014]; Mirza et al. [2011b]; Vasilescu et al. [2016a], we defined a task as a *well-defined work assignment with a specific goal that people divide their work into*, such as fixing a bug, or preparing for a team-meeting. A *task switch* occurs, when a person switches between two different tasks.

5.2.1 Task Switch Detection

Several researchers have explored the detection of task switches mostly for general knowledge workers. These approaches mainly differ in the features they used to identify the task boundaries or switches, ranging from semantic features to temporal features, the method they use, unsupervised versus supervised, and the way they evaluated their approach. One of the most prominent approaches is by [Shen et al., 2009, 2007, 2006; Stumpf et al., 2005] that is mainly based on semantic features and supervised learning. They reused an approach, TaskTracer [Dragunov et al., 2005], that allows users to manually indicate the tasks

they are working on, and additionally tracks their application interactions in the background, including window titles. Based on the assumption that windows of the same task share common words in their titles, they create vectors from window titles and identify task switches based on a textual similarity measure using the users' previously declared tasks and supervised learning. After the first version, Shen et al. [2006] further improved their approach to reduce the number of false positives and to be able to predict task switches online [Shen et al., 2009, 2007]. Their evaluation is based on a small set of two users and counts a task switch as accurate if it falls within a 4 to 5 minute time window of a real switch, which is a very coarse measure, given the frequent task switching in today's environment that happen every few minutes [González and Mark, 2004; Meyer et al., 2014]. Based on the assumption that switches between windows of the same task occur more frequently in temporal proximity than to windows of a different task, Oliver et al. [2006] examined a temporal feature of window switches within a 5 minute time window in addition to semantic features and using an unsupervised approach. An evaluation based on 4h of a single participant, they achieved a precision of 0.49 and recall of 0.72. Researchers have also used other temporal features, in particular, the frequency of window events, to determine task switches. Under the assumption that users navigate between windows more frequently when they switch tasks, as opposed to during a task, Nair et al. [2005] developed a system that calculates window event frequency based on fixed 5 minute time windows. An evaluation with 6 participants resulted in an accuracy of 50%. Mirza et al. [2011a] relaxed the constraint of a fixed time window, used adjusted frequency averages and studied the various approaches with 10 graduate students. They found that their approach improved the accuracy and achieved an overall accuracy of 58%. Overall, previous research has shown that detecting task switches is difficult, even for very short periods of time and in controlled environments. In our work, we focus on software development work and extend these approaches by including and examining both, semantic and temporal features of window events as well as user input features, and by conducting two studies with professional software developers.

Only little research has been performed on task switch detection in the software development domain and all of this research has focused solely on software development tasks within the IDE. As one of the first, Robillard and Murphy [2004] proposed to use program navigation logs to infer development tasks and they built a prototype, however, without evaluating it. In 2008, Coman and Sillitti focused on splitting development sessions into task-related subsections based on temporal features of developers' access to source code methods and evaluated their approach with 3 participants over 70 minutes each, finding that they can get close to detecting the number of task switches, yet the point in time when the task happens is a lot more difficult [Coman and Sillitti, 2008]. Zou and Godfrey [2012] replicated Coman and Sillitti's study in an industrial setting with six professional developers and found that the algorithm detects many more task switches than the ones self-reported by the participants with an error of more than 70%. Finally, on a more fine-grained level, Kevic and Fritz examined the detection of activity switches and types within a change task using semantic, temporal and structural features. In two studies with 21 participants, they found that activity switches as well as the six self-identified activity types can be predicted with more than 75% accuracy [Kevic and Fritz, 2017]. Different to these approaches, we focus on all tasks a developer works on during a day, not just the change tasks within the IDE.

5.2.2 Task Type Detection

Researchers also examined detecting the type of task or activity a person is working on. Most similar to our approach for task type detection is Koldijk et al. [2012]. They investigated the use of features over a fixed 5 minute time window, using mouse and keyboard input, application (switches) and the time of day. They tried to predict one of 12 task types that they identified in a survey, *e.g.*, read email, write email, plan, program, search information, and create visualization. The results of a field study with 11 researchers and an average of 10 hours of data per participant shows that the prediction is very individual and that a general classifier does not work well. Mirza et al. [2015] focused on classifying users' desktop interactions into six higher level activity types: writing, reading,

communicating, web browsing, system browsing and miscellaneous. They used temporal, interaction-based (application window events), and semantic features calculated over a 5 minute time window. In a 6 hour field study with five participants and a controlled lab study, they found that they can predict the activity category for each of these 5 minute windows with high accuracy (81%) and that interaction-based features work best. Researchers have also explored biometric features, such as Hassib et al. [2017] who used Electroencephalography to classify the task type according to cognitive load, but without looking at specific task types. In our work, we extend these approaches by not fixating on fixed time windows of 5 minutes but actually detecting the switches and by evaluating them with professional developers in the field.

We have been able to find only very little research on predicting task types for software developers. To the best of our knowledge, the only approach that is similar to our work is by Bao et al. [2018]. In their work, they automatically track low-level computer interaction data (user input and application usage) and use a Condition Random Field (CRF) based approach to segment the data and infer one of six development activities–coding, debugging, testing, navigation, search, or documentation–similar to our task types. An analysis of data collected from ten developers over a week shows that CRF is able to classify the activity with 73% accuracy. We extend the approach by focusing on all activities developers perform during their workday, not just development, and by examining when developers switch between different tasks.

5.2.3 Task Support

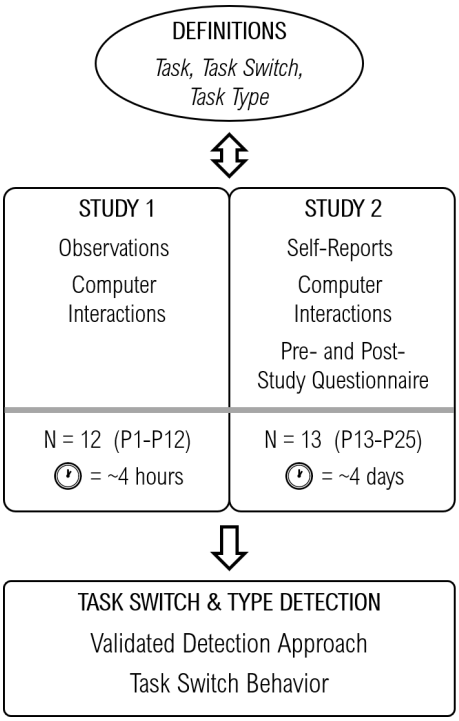
While there is a vast number of approaches to support specific development activities, such as code search, code review or debugging, only little research has looked into supporting developers with understanding and managing their tasks and the frequent switches between them. Several researchers have therefore proposed to explicitly model development tasks and to capture task contexts–artifacts and applications relevant to a task–to support developers in their task work, in particular by recommending relevant artifacts [Kersten and Murphy, 2006; Sahm and Maalej, 2010], identifying related tasks [Maalej et al., 2017],

easing the resumption of interrupted tasks and switching between them [Bragdon et al., 2010a,b; Card and Henderson Jr, 1986; Dragunov et al., 2005; Kersten and Murphy, 2006; Oliver et al., 2008; Rattenbury and Canny, 2007; Robertson et al., 2000; Smith et al., 2003], or scoping queries and recommending workflow improvements [Murphy et al., 2005]. Early approaches to support task switching, such as virtual workspaces [Card and Henderson Jr, 1986] or the GroupBar [Smith et al., 2003], provide interfaces that allow the user to manually group artifacts and applications with respect to tasks. The approaches Mylyn by Kersten and Murphy [2006], and TaskTracer by Dragunov et al. [2005], both explicitly capture task context by automatically recording user interactions within the IDE or the desktop environment respectively, given the user manually indicates the start and end of a task. While several of these approaches have great potential to support developers in their task work, they require some form of *manual* interaction to identify the task boundaries, something that developers often forget to do after using such a tool for a while [Kersten and Murphy, 2006]. Researchers have therefore examined how to best aid developers in identifying task boundaries retrospectively [Safer and Murphy, 2007], looked into more lightweight approaches for supporting task resumption through cues and without specific task context [Parnin and DeLine, 2010b], or explored the *automatic* mining of task contexts to support window switching [Oliver et al., 2008] and grouping files [Rattenbury and Canny, 2007]. Overall, an automatic and real-time task switch detection has thereby the potential to complement and significantly improve the value of most of these existing approaches for developers.

5.3 Study Design

To investigate the use of computer interaction data for predicting task switches and types, we conducted two field studies, a 4-hour observational study and a multi-day study with experience sampling, with a total of 31 professional software developers initially. The observations and self-reports served as the ground truth of participants' task switches and types, while we additionally gathered computer interaction data to extract features for our predictions. In both studies, we used

Figure 5.1: Overview of study design and outcomes.



the same definitions of tasks, task switches and types which we also shared with the participants. A brief overview of our study design is presented in Figure 5.1.

5.3.1 Study 1 – Observations

In our first study, we observed the work of 12 participants over a period of 4 hours to gather a richer understanding of developers' task switches and types they work on.

Procedure. For the observations, the observer, either the first or second author, followed a detailed protocol that we developed before the study. The very first observation session was performed by both observers at the same time. A cross-check of the two observation logs showed an inter-rater agreement of 97%, suggesting a high overlap of observing the same tasks and task switches.

Before each observation session, the observer explained the study purpose and process to the participants and asked them to sign a consent form, to install a

monitoring tool that tracks participants' computer interaction, and to describe the tasks they were planning to work on during the observation. The observer also introduced herself to nearby colleagues and asked them to ignore her as much as possible, and collaborate with the observed participant as they would normally do. After that, the observer placed herself behind the participant to prevent distractions, while still being able to see the screen contents on the participant's computer. Finally, the observer started the actual observation session and asked the participant to continue their work as usual.

We observed participants for a total of four hours each on a single workday: two hours before and two after lunch. For the observations, we followed the protocol of a structured observation session by Mintzberg [1980]. The observer wrote in an observation log [1] each time the participant switched from one task to another. Each entry in the observation log consists of a timestamp, a description of the reason for the task switch and a description of the task itself. We inferred tasks and their details from the active programs and their contents on the screen, as well as discussions participants had with co-workers. After each session, the observer validated the observed tasks and task switches with the participant, by going through the list of observed tasks and accompanying notes and modifying mistakes made during the observation.

Participants. We recruited 14 participants through professional and personal contacts from two large-sized and one medium-sized software companies. We excluded two participants for which we were not able to observe a sufficient amount of task switches (less than 10). Of the remaining 12 participants, 1 was female and 11 were male. Throughout the paper, we refer to these participants as P1 to P12. Our participants had an average of 10.8 ($\pm$7.4, ranging from 1 to 20) years of professional software development experience and were working in different roles: 8 participants identified themselves as individual contributors and 4 as developers in a leading position. All participants resided either in Canada or the United States.

[1] We used our own observation logging tool: `https://github.com/casaout/ObservationStudyTool`

Monitoring Tool. To collect computer interaction data from developers, we developed and used our own monitoring tool, PersonalAnalytics [2], for the Windows operating system. The tool tracks participants' mouse and keyboard interaction, as well as their application usage. For the mouse, the tool tracks the clicks (coordinates and button), the movement (coordinates and moved distance in pixels), and the scrolling (coordinates and scrolled distance in pixels) along with the corresponding time-stamp. For the keyboard, the tool records the type of each keystroke (regular, navigating, or backspace/delete key) along with the corresponding time-stamp. For privacy reasons, we did not record specific keystrokes. Our tool further records the currently active application, along with the process name, window title, and time-stamp whenever the window title changed or the user switched to another application.

Task Type Inference. We inferred task type categories by performing a Thematic Analysis [Braun and Clarke, 2006] on the basis of related work and our observation logs. The analysis process included first familiarizing ourselves with the observed task switches, open coding the observed and participant-validated tasks and accompanying notes, identifying themes, and categorizing the resulting themes into higher level task types. This process resulted in nine task type categories: *Development*, *Personal*, *Awareness & team*, *Administrative*, *Planned meeting*, *Unplanned meeting*, *Planning*, *Other* and *Study*. The task types are described in more detail in Table 5.4 and discussed in Section 5.6.1.

5.3.2 Study 2 – Self-Reports

To capture a longer time period and more breadth in developers' work, we conducted a second field study with 13 participants over a period of 4 workdays each. For this study, we used experience sampling, in particular we regularly prompted participants to self-report task switches and types. By using experience sampling, we also wanted to mitigate the risk of a bias in participants' behavior

[2] `https://github.com/sealuzh/PersonalAnalytics`. Details can be found in [Meyer et al., 2017b].

due to an observer sitting behind them, which, for example, could lead to participants being less likely to browse work unrelated websites.

Procedure. Before the study, we emailed participants a document explaining the study goal and procedure, asked them to sign a consent form and to answer a pre-study questionnaire with questions on demographics, their definition of a task, reasons for switching between tasks, and on the task types they are usually working on. After that, participants were asked to install the same **monitoring tool** that we described above on their main computer. In case participants worked on multiple computers (*e.g.*, a desktop and a laptop), we asked them to install the monitoring tool on both devices. Participants were further asked to read our definitions of a task, task switch and task type, as well as instructions on how to use the **self-reporting component** that we added to our monitoring tool. Finally, participants were asked to pursue their work as usual for the next couple of workdays while also self-reporting their task switches and types when the pop-ups/prompts appeared.

For this study, our tool prompted participants once per hour to self-report their task switches and types for the previous hour. The self-reporting step is explained in more detail below. We intentionally decided to use an interval of one hour rather than a full day, to balance the intrusiveness of the prompts with the ability to accurately remember tasks and task switches over the previous time interval [Tourangeau et al., 2000]. To further ensure high quality in the collected self-report data we further allowed participants to withdraw from the study at any point in time, and to pick the time for their participation themselves. In addition, and to avoid boredom or fatigue, we asked participants to respond to a total of 12 to 15 prompts, assuming an average of four self-reports per day and a total of three to four workdays for participation. This number was a result of several test-runs over multiple weeks and from qualitative feedback gathered with a pilot participant, a professional developer. Furthermore, we provided support to postpone self-report prompts for 5 minutes, 15 minutes, or 6 hours, and built and refined the self-reporting component to require as little effort as possible to answer, *e.g.*, by letting participants answer the required fields by simply clicking

on elements instead of asking them for textual input. Finally, each pop-up also asked participants to report their confidence with their self-reports.

Throughout the study, participants could check the number of completed pop-ups. Once they completed 12 pop-ups, participants could notify us and upload the collected data and self-reports to our server. The upload wizard once again described the data collected and allowed participants to obfuscate the data before sharing it with us. At the end of the study, participants were asked to answer a post-study questionnaire with questions on the experienced difficulties when self-reporting task switches and task types, on further task types they were working on, and on how they could imagine using information on task switches and types. After completing the survey, participants were given two 10 US$ meal cards to compensate for their efforts.

Participants. We recruited 17 participants through professional and personal contacts from one large-sized software company. We discarded data from three participants that self-reported less than 10 task switches in the days of their participation. We further discarded the data of one participant whose definition of a task switch was very different to ours and the rest of the participants (*i.e.*, he considered every application switch a task switch). Of the remaining 13 participants that we used for the analysis, 2 were female and 11 were male. Our participants had an average of 12.1 ($\pm$8.2, ranging from 1 to 30) years of professional software development experience and were working in different roles: 10 identified themselves as individual contributors and 3 as developers in a leading position (*i.e.*, Lead or Manager). All participants resided in the United States. In the paper, we refer to these participants as P13 to P25.

Self-Reporting Component. The self-reporting component is part of our monitoring tool and includes a pop-up with three pages. The *first* page asked participants to self-report the task switches they experienced in the past hour. It visualized participants' application usage on a timeline using different colors for each application and allowed them to self-report their task switches by clicking on the lines denoting applications switches. We restricted the task switch self-reports

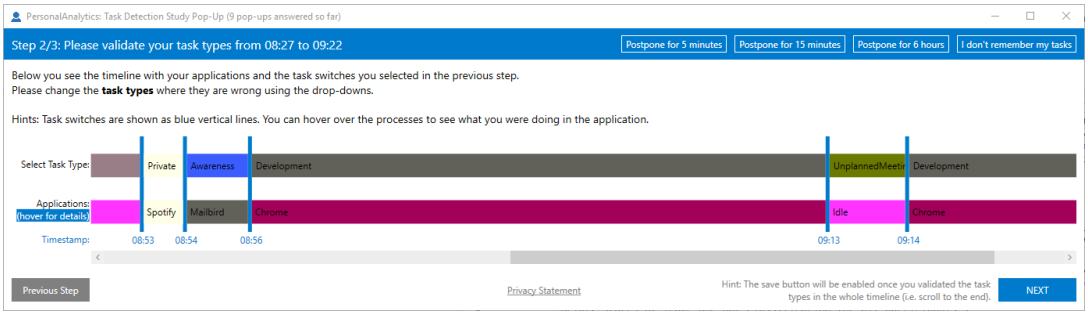


Figure 5.2: Screenshot of the second page of the experience sampling pop-up that asked participants to self-report their task types.

to a granularity of application switches with a minimum length of 10 seconds for a variety of reasons: First, we assumed that most of participants' task switches coincide with application switches (*e.g.*, switching from the email client to the IDE, or from the browser to an IM client) and fewer happen during a session uniquely spent within the same application (*e.g.*, switching tasks directly in the IDE or in the browser). And, we wanted to avoid cluttering the user interface of our self-reporting component and simplify the reporting for participants. Similar to Shen et al. [2009], the timeline visualization provided additional details when the participant hovers over an application, such as the application name, time when it was used, window title(s) and user input produced in that application. As soon as participants completed self-reporting their task switches for the whole previous hour, they could proceed to the *second* page and self-report their task types (see Figure 5.2). On the second page, we visualized the same timeline as before, but added another row that prompted participants to select task types from a drop-down menu. After selecting the task types for all task segments, participants could proceed to the last page. The *third* page asked participants to self-report their confidence with their self-reports of task switches and task types on a 5-point Likert-scale (5: very confident, 1: not at all confident) and optionally add a comment. The user interface we used to collect the ground truth for task switches and types resembles the one by Mirza et al. [2011a, 2015, 2011b]. The supplementary material[3] includes the pre- and post-study questionnaires and additional screenshots detailing the self-reporting component.

[3]Supplementary material: `http://doi.org/10.5281/zenodo.3244076`

5.4 Data and Analysis

For this study, we collected two rich data sets, including observed or self-reported ground truth data, and automatically tracked computer interaction data. Prior to the main analysis of the data, we performed multiple pre-processing steps, including data segmentation and feature extraction, which are summarized in the remainder of this section.

5.4.1 Collected Data

For our **study 1**, we collected observation logs for a total of 51.7 hours of work and an average of 4.3 (±1.3) hours per participant. For our **study 2**, we collected self-reports for a total of 58 workdays and an average of 4.5 (±1.7) days per person. On average, participants reported a high confidence with their self-reports (>3) in 20.6 (±9.0), and a medium or low confidence (≤3) in 22.2 (±16.7) of the pop-ups they answered. For our analysis, we decided to only use the data of the 268 self-reports with a high confidence (>3), accounting for a total of 268 hours of work, and discarding the rest (289 self-reports). Table 5.1 reports statistics on the self-reports. Since overall, only 11% of the pop-ups were postponed by participants, one reason for the relatively high number of self-reports with medium or low confidence could be that the pop-ups appeared at inopportune moments and participants did not remember they could postpone it. Instead, participants might have just clicked through the pop-up and reported a low confidence to not distort the data. We discuss possible threats in Section 5.7 and improvements in Section 5.8.

5.4.2 Time Window Segmentation

To calculate and extract **task switch detection** features, we defined the time windows to be between application switches, which we call *application segments*. We chose to use application segments, since developers on average spend only 1 to 2 minutes in an application before switching to another application, and possibly, switching tasks [González and Mark, 2004; Meyer et al., 2017a, 2014]. In contrast,

Table 5.1: Self-reports from study 2.

	All	per Participant
Days participated	58	4.5 (±1.7)
Pop-ups displayed to participants	557	42.8 (±21.6)
Pop-ups answered by participants	268	20.6 (±9.0)
- Pop-ups answered within 5 minutes	158	12.2 (±6.3)
- Pop-ups answered after 5 minutes	110	8.5 (±5.4)
Pop-ups postponed by participants	62	4.8 (±3.5)
Pop-ups discarded by researchers	289	22.2 (±16.7)

previous approaches predominantly used longer and fixed window lengths of 5 or 10 minutes [Nair et al., 2005; Oliver et al., 2006; Shen et al., 2009, 2007]. These shorter and more flexible time windows at borders of application switches allow to more accurately capture developers' behaviors, and to more precisely locate the point in time of the task switch. For the **task type detection** features, we used the time windows between two task switches (as identified by the self-reports or observations), which we call *task segments* for the feature extraction.

5.4.3 Task Switch Features Extracted

A next step towards building a classifier for task switch detection is to extract meaningful features from the raw computer interaction data collected by the monitoring tool. Hence, we developed features based on the heuristics participants stated as indicative of their task switches in the post-questionnaire of study 2, adapted features that have been linked to developers' task switching behavior in prior work, as well as features based on our own heuristics. The features we used are described in Table 5.2.

Since task switch detection is a special case of change-point detection [Basseville et al., 1993; Gustafsson and Gustafsson, 2000], which is trying to detect abrupt changes in time-series data, many of our features capture the similarity between characteristics of the previous application segments and the current one, such as the difference in the average number of keystrokes. However, it is not yet clear, how many steps one should go back into the past for these comparisons.

Table 5.2: Features analyzed in our study and their importance for predicting task switches and task types.

Features	Import. Switch	Import. Type *All*	Import. Type *UI*
User Input Features	**48.2%**	**47.6%**	**77.0%**
Keystroke differences: difference in the number of navigate/backspace/normal keystrokes pressed per second between the previous and current application/task segment [Bao et al., 2018; Iqbal and Bailey, 2007; Koldijk et al., 2012; Mirza et al., 2015]	17.3%	17.5%	26.4%
Mouse click differences: difference in the number of left/right/other mouse clicks per second between the previous and current application/task segment [Bao et al., 2018; Iqbal and Bailey, 2007; Koldijk et al., 2012]	18.0%	13.9%	23.5%
Mouse moved distance: total moved distance (in pixels) of the mouse per second [Iqbal and Bailey, 2007]	7.6%	10.0%	17.1%
Mouse scrolled distance: total scrolled distance (in pixels) of the mouse per second [Bao et al., 2018; Koldijk et al., 2012]	5.3%	6.2%	10.0%
Application Category Features	**24.5%**	**39.1%**	**NA**
Switch to/from specific application category: switch to/from a specific application category (*e.g.*, messaging), while the previous one was different. Application categories considered: messaging [PS],[Iqbal and Bailey, 2008], idle [PS],[Maalej et al., 2017], code review [PS], utility, work unrelated browsing [PS], music [PS],[Bao et al., 2018; Iqbal and Bailey, 2008; Mirza et al., 2011a]	22.6%	NA	NA
Same application category: the current application category is the same as the one in the previous application segment, *e.g.*, both are messaging [Iqbal and Bailey, 2008; Mirza et al., 2011a]	1.9%	NA	NA
Time spent per application category: the percentage of the total duration of the task segment that was spent in each of the 17 application categories [Koldijk et al., 2012; Mirza et al., 2015; Shen et al., 2009]	NA	39.1%	NA
Switching Frequency Features	**18.5%**	**13.3%**	**23.0%**
Difference in the window switches frequency: difference of the number of switches between windows of the same or a different application per second between the current and the previous application/task segment [Koldijk et al., 2012; Oliver et al., 2006; Shen et al., 2009]	8.2%	13.3%	23.0%
Difference in the time spent in an application: difference of the total duration spent between the current and the previous application segment [Shen et al., 2009]	10.3%	NA	NA
Lexical Features	**8.8%**	**0%**	**0%**
Code in window title: the window titles of the current and previous application/task segments both contain code, as identified by text that is written in camelCase or snake_case. Can also distinguish between development and other file types	1.6%	0%	0%
Lexical similarity of the window titles and application names: cosine similarity based on the term frequency-inverse document frequency (TF-IDF) between the current and previous application segments' window titles or application names [Brdiczka et al., 2010; Oliver et al., 2006; Shen et al., 2009]	7.2%	NA	NA

References on these features (in blue) are either on previous related work or participants' suggestions (PS). A feature importance of NA denotes that the feature was not used for the prediction group. For the task type columns, 'All' denotes that all features were considered, 'UI' indicates that only the user interaction features were used, and the application category features were ignored.

Therefore, we analyzed how many steps (in our case application segments) one should go back for calculating features. In particular, we compared the precision and recall of the task switch detection taking into account 1 and up to 10 steps back into the past. Our analysis of the results indicated that after an initial increase of the precision for detecting a switch, the precision and recall gradually drop as the number of steps increases. We therefore chose 2 as the number of steps to go back in terms of application segments. As a result, the total number of features used for the task switch detection is 56, which is double the number of unique features used: once calculated for comparing the current with the previous application segment, and once to compare the previous two application segments. In the following, we provide an overview over the features used:

User Input Features. The first feature group are user input features based on keyboard and mouse interaction, such as the difference in the number of keystrokes the participant pressed per second between this and the previous time window segment.

Application Category Features. We categorized commonly used applications into one of 17 predefined application categories (based on our classification in previous work [Meyer et al., 2017a]). These include categories specific to software engineering, such as *Development*, *Code Review* or *Testing*, but also more general ones, such as *Read/Write Document*, *Email* and *Web Browser*. They are leveraged in 12 features that capture switches to or from a specific application category, such as switching to a messaging application or becoming idle. These features were selected based on participants' suggestions of what they consider to be good indicators for switching to another task. Since switching to another application might be another indication for a task switch [Iqbal and Bailey, 2008; Mirza et al., 2011a], we added one feature that captures these.

Switching Frequency Features. In the post-study questionnaire, participants mentioned that they often navigate through several applications to clean-up

their computer right before starting a new task, which is why we added a temporal feature based on the window switching frequency. One feature captures the difference in the time spent in an application, since this might be another indicator for a task switch, either because a switch is less likely immediately after a task switch, and the likelihood of a task switch increases as time passes [Shen et al., 2009].

Lexical Features. Inspired by prior work [Brdiczka et al., 2010; Oliver et al., 2006; Shen et al., 2009], we also added three lexical/semantic features that are extracted from application names and their window titles. Since window titles might include code snippets, such as a class or method name or development file type, we added a feature that captures whether the window title contains text written in `camelCase` or `underscore_case`, and whether this is different to the previous segment. To determine whether the previous and current application segments have a contextual similarity, two features are calculated based on the cosine similarity of the window titles and application names using the term frequency-inverse document frequency (TF-IDF). Note that the application name and window titles were also used to determine the application category features. In addition, and unlike some previous work, we explicitly did not capture file contents to reduce intrusiveness and avoid privacy concerns [Rattenbury and Canny, 2007; Soules and Ganger, 2005; Wu et al., 2005].

5.4.4 Task Type Features Extracted

For the task type detection, we reused the same features as in the task switch detection whenever possible. However, some features required adaption or made no sense in this context. First, as the time window for task type detection encompasses one or multiple application segments, we replaced the *application category* features with a feature that captures the ratio between the *time spent in the specific application category* and the time spent in the task segment. This allowed us to determine the dominant application category in a task segment. Second, we eliminated the lexical similarity features that relied on similarity measures between window titles of individual applications. This resulted in a total of 27 features used for the task type detection.

5.4.5 Outcome Measures

For the **task switch detection**, we labeled the application segments with the observed (study 1) and self-reported (study 2) task switches. In case the previous application segment was marked as a task switch, we labeled the current application segment with *Switch*, otherwise with *NoSwitch*. Therefore, our task switch detection approach is at most the duration of the application segment away from the actual task switch, which was an average of 1.6 minutes ($\pm$2.2) in our study. For the **task type detection**, we labeled each task segment with the observed or self-reported task type. Descriptive statistics regarding participants' task switching behavior and the task types they worked on can be found in Section 5.5.3 and Section 5.6.4, respectively.

5.4.6 Machine Learning Approach

We used scikit-learn by [Pedregosa et al., 2011b], a widely used machine learning library for Python, to predict task switches and task types. We evaluated several classifiers by applying them to our feature set and testing different parameter values. A RandomForest classifier with 500 estimators outperformed all other approaches, including a Gradient Boost-Classifier, Support Vector Machine (SVM), Neural Network and Hidden Naïve Bayes classifier. A RandomForest classifier is one form of ensemble learning that creates multiple decision tree classifiers and aggregates their predictions using a voting mechanism [Breiman, 2001; Liaw et al., 2002]. It does not require a pre-selection of features and can handle a large feature space that also contains correlated features. Hence, for the remainder of this paper, the presented results were obtained using a **RandomForest classifier**. Prior to classification, we impute missing values by replacing them with the mean and apply standardization of the features, which centers the data to 0 and scales the standard deviation to 1. These common steps in a machine learning pipeline can improve a classifier's performance [Pedregosa et al., 2011a]. For the **task switch detection**, we further apply Lemaître's implementation of SMOTE, which is a method for oversampling and can considerably boost a classifier's performance in the case of an imbalanced dataset such as ours [Lemaître et al.,

2017]. For the **task type detection**, where as much as 80-90% of the reported types are of the *Development* class we instead employ penalized classification to correct problems caused by class imbalance, as SMOTE has significant drawbacks when the minority classes have a limited number of samples [Nguyen et al., 2009].

We built both **individual and general models**, where an individual model is trained and tested with data solely from one participant and a general model is trained on data from all participants except one, and tested on the remaining one. Individual models often have a higher accuracy since they are trained on a person's unique behavioral patterns. On the other hand, general models are usually less accurate but have the advantage of solving the *cold-start-problem*, which means that no prior training phase is required and the model can be applied to new users immediately.

To evaluate the individual models, we applied a 10-fold cross-validation approach, where the model was iteratively tested on 1/10 of the dataset while being trained on the remaining data. We adapted the cross-validation approach to account for the temporal dependency of the samples. In particular, there is a dependency between samples in close temporal proximity, since data from the preceding samples is incorporated in the features. To ensure a valid and realistic evaluation of the model, we therefore deleted h samples on either side of the test set block [Racine, 2000]. In our case, we chose h=10 since we included up to 10 preceding samples in the feature calculation.

5.5 Results: Detecting Task Switches

5.5.1 Task Switch Detection Accuracy

Table 5.3 gives an overview of the task switch detection performance of individual and general models. We split the presentation of the data into the two studies, since they were collected with a different method. Overall, our analysis revealed that we can detect task switches at their exact location with a high averaged accuracy of 87% (precision: 67% and recall: 37%) when trained with individual models. The performance of the classifier drops considerably when we apply the

Table 5.3: Overview of the performance of the task switch detection, for both individual and general models.

	INDIVIDUAL MODELS					GENERAL MODEL				
		Switches		No Switches			Switches		No Switches	
Dataset	**Acc.**	**Prec.**	**Rec.**	**Precision**	**Rec.**	**Acc.**	**Prec.**	**Rec.**	**Prec.**	**Rec.**
Study 1: Observations	83%	59%	40%	86%	92%	68%	46%	62%	88%	73%
Study 2: Self-Reports	89%	69%	36%	90%	97%	71%	43%	58%	90%	75%
All	**87%**	**67%**	**37%**	**89%**	**96%**	**73%**	**46%**	**55%**	**90%**	**79%**

general model, to an averaged accuracy of 73% (precision: 46% and recall: 55%). For the individual models, we compared the results of each participant's model (see supplementary material). It reveals that the prediction performance varies quite substantially for each participant. As expected, both the individual and general models of *study 2* performed slightly better than the ones in *study 1*.

We also analyzed the accuracy of the task switch detection of our individual models more qualitatively by calculating the **error distance** between each predicted and the actual task switch. The average error distance is 1.39 (±3.36) application-switches, meaning that in case of a faulty task switch detection, the actual task switch was 1.39 applications before or after the predicted one. 14.3% of the task switches our model predicted have a distance of 1 application-switch, 5.0% have a distance of 2 application-switches, and 20.7% have a distance of 2 or more application-switches.

5.5.2 Task Switch Feature Evaluation

A Random Forest classifier can deal well with a larger number of features which makes prior feature dimensionality reduction of our 56 features obsolete [Breiman, 2001; Liaw et al., 2002]. While we do not apply a feature selection technique in our approach since it would only select the most predictive features in the model, we are still interested in learning if certain features are generally more important, especially across different participants. The second column of Table 5.2 contains the feature importance as attributed by the RandomForest classifier using all features and averaged over all participants' individual models. To calculate the feature importance metrics, we used the Gini impurity measure from scikit-

learn, which captures the feature's ability to avoid mis-classification [Pedregosa et al., 2011b]. The most predictive feature groups are user input (48.2%) and application category (24.5%). The feature group with the least predictive power are the lexical features (8.8%). The supplementary material includes the feature importances of each individual feature.

5.5.3 Descriptive Statistics of the Dataset

Participants switched frequently between tasks, with a mean task switch rate of 6.0 (±3.7, min: 1.8, max: 18.9) times per hour. The average time spent on each task was 13.2 (± 7.3, min: 3.1, max: 30.8) minutes [4]. Developers' task switch behaviors are similar to previous work [González and Mark, 2004; Meyer et al., 2014].

5.6 Results: Detecting Task Types

5.6.1 Identified Task Type Categories

As described in more detail in Section 5.3, we inferred task type categories after collecting task and task switch data from observing 12 developers at work and performing a Thematic Analysis. This resulted in nine task type categories we described in Table 5.4. In the post-study questionnaires of study 2, participants reported that they agreed with the identified task types and generally had no issues to assign them. However, two participants mentioned that a task type for *Support* duties was missing:

> "*[Support]-Duties. These are very specific tasks that require a lot of different things to do. It's not Development and it can be a lot of ad-hoc and requires many context switches.*" - P14

Two participants mentioned that it was sometimes difficult to know if time spent on emails should be assigned to *Development* or *Awareness & team*:

[4] We do not report individual results for the two studies, since the task switch rate (p-value=.056) and time spent on a task (p-value=.215) are not significantly different in the two datasets.

> "*I was sometimes unsure of how to classify the time I spent responding to emails. I generally classified it as development since most of the emails were development-related.*" - P21

Most of our task type categories are consistent with previous work that investigated knowledge workers' tasks [Czerwinski et al., 2004; Kim and Choe, 2019; Koldijk et al., 2012; Reinhardt et al., 2011]. For example, *Meetings*, *Administrative*, *Planning* and *Private* were also prevalent in work by both Czerwinski et al. [2004] and Kim and Choe [2019]. The latter further divided project work (in our case *Development* tasks) into *Documenting and Conceptualizing Ideas*, *Environment and Development* and *Design*. We did not make these finer-granular distinctions since we did not want to make the self-reporting of task types in the second study too complicated, which would degrade the quality of self-reports.

5.6.2 Task Type Detection Accuracy

Table 5.4 shows the results of our task type detection approach across all 9 task type categories. We omit the accuracy metric in this table, as recall is a measure of individual class accuracy, and since the recall presented in the *all* row is weighted by class size it therefore assumes exactly the same value as accuracy. As with the task switch detection analysis, we trained both individual models and one general model which was trained on all participants. To save space, precision statistics are not shown for the general model, but they were observed to be lower across the board as compared to those values seen in the individual models. The *Administrative* task type was not predicted a single time by the general classifier, and as thus the precision scores were undefined for this class. In general, the individual models (precision 59%, recall 61%) outperformed the general model by a large margin (precision 44%, recall 50%).

Table 5.4: Overview and descriptions of the task type categories, the average time developers spent on each task type per hour, and the performance of our task type detection approach, for both individual and general models.

			INDIVIDUAL MODELS				GENERAL MODEL	
	Avg (Stdev)	Sample	All Features		UI Features		All Feat.	UI Feat.
Task Type Category	**mins/h**	**Size**	**Prec.**	**Rec.**	**Prec.**	**Rec.**	**Rec.**	**Rec.**
Development: bug-fix, refactoring, code review, implementing new feature, reading/understanding documentation/code, testing, version control, dev.-related learning	37.2 (±12.2)	612	70%	85%	62%	77%	77%	78%
Personal: work unrelated web browsing, private emails or texts, (bio or lunch) break	9.7 (±7.0)	170	48%	45%	42%	34%	32%	23%
Awareness & team: reading/writing emails, discussions/answering questions in IM	5.3 (±6.0)	234	64%	53%	40%	35%	44%	15%
Administrative: often routine tasks, *e.g.*, reporting work-time, expenses report, paperwork	4.0 (±3.6)	12	50%	17%	33%	8%	0%	0%
Planned Meeting: attending a scheduled meeting/-call, *e.g.*, weekly scrum, weekly planning meeting	3.6 (±2.7)	94	40%	40%	33%	31%	4%	1%
Unplanned Meeting: attending an ad-hoc, informal meeting, usually with one team-member only, *e.g.*, unscheduled phone call, colleague asking a question	3.1 (±2.8)	90	43%	29%	36%	29%	25%	18%
Planning: in the calendar, task list, work item tracker	3.0 (±3.5)	90	31%	24%	26%	16%	1%	0%
Other: tasks that do not fit into the other categories. Participants mentioned that these were support-duty, document writing (*e.g.*, in PowerPoint, Word) and for product development/innovation.	2.7 (±3.9)	40	47%	25%	3.7%	2.5%	0%	1%
Study: work related to this study (*e.g.*, talking to observer, filling out questionnaire)	1.9 (±1.9)	64	67%	58%	49%	41%	69%	20%
All		**1406**	**59%**	**61%**	**46%**	**49%**	**50%**	**41%**

*The 'All Features' columns show results using models trained with all features, while the 'UI Features' columns show results from models trained using only user interaction features (*i.e.*, excluding application category features).*

One important aspect of our approach that distinguishes it from previous work (*e.g.*, [Bao et al., 2018; Iqbal and Bailey, 2008; Mirza et al., 2011a]) is its ability to make predictions even on previously unseen applications. To demonstrate this, we split the results into two categories: with the manual application category mappings (*All Features*) and without (*UI Features*). The *UI Features* include all user interaction features, but exclude application features. While the combined approach proved to be superior, user input features still proved to have high predictive power on their own. Overall, there was a 28.2% increase in precision when including the application category features, and a 24.5% increase in recall.

We also found there was a substantial difference in performance depending on the task type category. The *Development* task type proved to be the easiest to predict, achieving high recall (85%) and precision (70%) scores. Conversely, the *Planning* task type saw very poor results, with only 24% recall and 31% precision. These results are somewhat in line with what one might expect. Naturally, some task categories are more difficult to predict than others. For instance, discerning the nature of a meeting (planned or unplanned) based purely on a users applications used and input activity seems to be nearly impossible. In some categories, especially in both of the meeting categories and the private category, there was a large amount of idle time recorded for many participants, which also partially contributes to the confusion seen in the results.

5.6.3 Task Type Feature Evaluation

The third and fourth column of Table 5.2 show the Gini feature importances we calculated for our RandomForest classifiers, averaged over all participants. When considering *All Features*, we found the *time spent per application category* features to have by far the greatest importance (39.1%), followed by keystroke features (17.5%). However, the combined *user input feature group* contributed more than any other feature group (47.6%). The lexical features did not contribute at all to the results of the classifier, which suggests there is room for improvement in this area as window titles can contain a substantial amount of hints that could help to identify a specific task. The supplementary material includes individual task type feature importances.

5.6.4 Descriptive Statistics of the Dataset

On average, developers worked on 6.1 (±1.6, min: 3, max: 9) different task types during the studied time periods, indicating that most of the identified task types are relevant to developers. The task type participants self-reported having spent the most time on is *Development*, with an average of 37 (± 12) minutes spent per hour. Table 5.4 reports details for all task types as well as the number of participants who self-reported having worked on the task type. We also analyzed if having a higher diversity in work (*i.e.*, working more different task types) correlates with developers switching more between tasks. There is a weak positive correlation (Pearson's $r = .0.32, p = .12$).

5.7 Discussion

We discuss implications of our results, possible improvements and practical applications of automated task detections.

5.7.1 Improving Task Switch and Type Detection

We found that for both task switch and type detection, the **individual models outperform the general model**, with an overall accuracy of 87% compared to 73% for switch prediction, 61% to 50% for type prediction. Even though we collected data from a rather large sample of 25 participants (compared to similar work), we were not yet able to build reliable general models, which could solve the cold-start-problem. The general models' inability to discover common patterns across all participants emphasizes how individual and diverse developers' task switch behaviors are. More research is required to explore reasons for and better balance these individual differences. For example, we could imagine to include information about a developer's personality and company culture to train a classifier that works well for developers with similar work habits, instead of building a general one for everyone. Future work could also study the predictive power of features extracted from additional data sources, such as emails, meetings, biometrics (*e.g.*, detecting when a user is away from the computer), and more detailed development related data (*e.g.*, activities inside the IDE).

The relatively low feature importances of our lexical features shows further potential to more effectively leverage contextual information. Besides calculating lexical similarity based on cosine similarity (TF-IDF) of window titles, we also experimented with variations, such as word embeddings or using term-frequencies without weights. They led to even less predictive features, which is why we did not report them separately. One reason could be the little overlap in the window title data. While leveraging additional lexical data (*e.g.*,, file contents) could bear a lot of potential, the privacy concerns for this data were too high for participants.

Ideally, a task switch and type detection would be very close to real-time, *i.e.*, close to the exact time a switch occurs. With our approach, there can be a **prediction delay** of a maximum of one unique application segment, on average 1.6 minutes (±2.2), when predicting a task switch. This delay is considerably smaller compared to previous approaches that applied fixed window lengths of (usually) 5 minutes (*e.g.*, Koldijk et al. [2012]; Mirza et al. [2015]; Nair et al. [2005]; Oliver et al. [2006]; Shen et al. [2009, 2007]). Nonetheless, future work could further reduce the prediction delay by further shortening the smallest possible segment size, in our case application switches. This would allow to also identify switches within an application, such as when a developer is switching tasks inside the web browser.

Our approach extends prior work by combining temporal and semantic features, by developing new features, and by not being limited to capturing task switches and types within the IDE only. The evaluation of our approach in a field-study with 25 professional developers, compared to 1 to 11 participants in previous work, revealed higher accuracy and less delay in the predictions than comparable prior work.

5.7.2 Applications for Automated Task Detection

An active area of research aims to better support developers' frequent task switching, for example by supporting resuming interrupted tasks or by easing task switching (see Section 5.2.3). So far, most approaches are limited to developers' *manual* identification of task switches, and their evaluations have

pointed out challenges this poses for developers. Our approach demonstrates the feasibility of *automatically* detecting task switches and types in the field, based on a few hours of training data, which makes it possible to increase the value of previous approaches significantly and stimulate new research and tool support. In the post-study questionnaire of study 2, participants described concrete applications that we qualitatively analyzed and related to prior work, which resulted in the following three main opportunities for applying automated task detection:

One application of an almost real-time detection of task switches that 8 (out of 13) participants described is to **actively reduce task switching**. This includes automatically blocking notifications from email, instant messaging or social networks when a developer is focused on a (challenging) task, to allow extended times of deep focus:

> "*What if Windows has a built-in and personalized model about when to give you notifications. I feel like there is a good middle ground between forcing the user to turn off notifications from the OS and having too many notifications interrupting the user.*" - P25

Reducing task switching at times of high focus could greatly reduce multi-tasking, a major source of stress and quality issues [González and Mark, 2004; Mark et al., 2018, 2008b, 2017]. Similarly, an automated task switch detection could improve interruptibility classifiers and postpone in-person interruptions from co-workers to task switch borders, times they are less costly [Fogarty et al., 2005; Züger et al., 2017; Züger et al., 2018].

Another application of automated task detection could be to support the **resumption of suspended or interrupted tasks**. Participants did not suggest this application themselves, but 8 (out of 13) rated it as 'useful' or 'very useful' in a follow-up question of the final questionnaire. According to Parnin and Rugaber [2009], a major challenge of task resumption is to rebuild the interrupted task's context. Our task detection approach could *automatically* identify and build up task contexts, consisting of work artifacts (*e.g.*, websites or files) and applications that the developer used for the task. Applying similar summarization approaches as seen in other areas of software development [Nazar et al., 2016; Treude et al.,

2015] could be presented to the user as cues upon returning to the suspended task, which has been shown to considerably reduce the resumption lag [Altmann and Trafton, 2004; Bailey et al., 2001; Rule et al., 2015]. While previous approaches, such as TaskTracer [Dragunov et al., 2005], Scalable Fabric [Robertson et al., 2004], GroupBar [Smith et al., 2003] and Mylyn [Kersten and Murphy, 2006], allow the capturing and presentation of task context, they require the user to *manually* group related artifacts or manually state the start and end of a task, thus, reducing chances of long-term adoption. Even tough there is room for improvement as discussed above, our approach can serve as a starting point to automate these approaches, since it can already be beneficial to receive help with resuming some tasks, as long as they are detected correctly.

A third opportunity of application that 10 (out of 13) participants suggested is to use *automated* task detection to **increase their awareness about task work** and time spent on tasks, which could help to identify opportunities for work habit and productivity improvements. This is in line with a survey with 379 developers that showed the most-often mentioned measurement of interest when reflecting about productivity are the tasks developers made progress on and completed in a workday [Meyer et al., 2014]. An aggregated visualization of the automatically inferred tasks could give developers insights such as how much time they spend on different tasks, when they worked on planned versus unplanned tasks, or their multi-tasking behaviors:

> "*It can help point out different working styles that are also effective and efficient. Not everyone works in the same way.*" - P24

Recently, researchers started building retrospective dashboards for developers [Beller et al., 2016; Codealike, 2019; Meyer et al., 2017b; Wakatime, 2019] and other knowledge workers [Kim et al., 2016; RescueTime, 2019; Whittaker et al., 2016], usually by visualizing data on the level of applications or application categories, but suggesting that a per-task level would be more beneficial. An increased awareness about one's task switching behavior could support developers to identify goals that help to maintain and improve good work habits, such as reducing multi-tasking or actively blocking notifications from distracting services and websites at times they need to focus. Participants further suggested that the

data could help to reduce administrative workloads that require them to report time spent at work:

> "*We're often asked to report at the end of the month how much time we spent on support requests (...) versus development work. That kind of info is tedious to track manually, but a tool could generate an automatic report as needed, allowing for more accurate counts.*" - P22

Lu et al. [2018] recently showed that the lack of logs of activities and tasks is often a hindrance to be able to transfer them into time reports. While a few time-tracking tools already exist (*e.g.*, DeskTime [2019], TimeDoctor [2019]), they all require users to manually specify the start and end of a task.

5.8 Threats to Validity

Observing Developers in Study 1. The internal validity of our results might be threatened by the presence of the observers during the observation sessions, causing developers to diverge from their regular work habits, *e.g.*, having less breaks than usual. Observing participants on a single day only might not be representative of the participant's regular workday. We tried to mitigate these risks by not interacting with participants during the observations, splitting up the session into two two-hour blocks, sitting as far away from the participant as possible, telling co-workers beforehand that they could still communicate and interrupt as usual, and by allowing the participant to pick an optimal timeslot that is representative of their usual work. Our observational study has the advantage that, rather than performing a lab study or experimental exercise, participants were observed during their real-world work, thus increasing generalizability and realism. However, the above mentioned risks of observing developers at their workplace make it very difficult to scale observational studies and observe them over many days. Hence, we did not rely only on observations, but also on participants' self-reports and with that, combining two methods and strengthening our overall approach.

Self-Reporting in Study 2. While collecting participants' task data using self-reports has proven to be a valuable approach to scale the collection of labeled data for supervised learning, there are a few limitations. First, we rely on the accuracy of participants' self-reports. For example, they might not always have been able to accurately remember their tasks, or filling out the pop-up regularly might be perceived as cumbersome after a while. In Section 5.3.2, we describe our actions to minimize these risks in detail, including the ability to postpone a pop-up and collecting confidence ratings. Aiming to make the self-reporting as easy as possible required limiting the segmentation of the self-reports to applications and excluding application switches shorter than 10 seconds. Future work could investigate how to give participants good-enough cues that allow them to accurately self-report switches within applications (*e.g.*, switching from a news website to the work item tracker in the browser) without making the interface too cluttered. Finally, the reliance on collecting computer interaction data only, instead of also including other sensors such as heart-rate monitors or cameras, limits our knowledge of what is happening when there is no input to the computer, *e.g.*, in the case of idle times from using the smartphone, reading a document without scrolling, or a discussion with a co-worker.

Sample Size. A further threat to the external validity of our results could be the number of participants. A higher number of participants might have led to a more robust general model to predict task switches and task types. Nonetheless, collecting task data from 25 participants is considerably higher than what was reported in previous work (between 1 and 11 participants). We tried to mitigate this threat by selecting participants from four different software companies in various locations.

Task Definitions. The construct validity of our results might be threatened by our definitions of a task (switch) and our open coding approach to identify task type categories. To minimize this risk, we based our definitions of task, task switch and task type on previous work, and asked participants about their own definitions in both studies (Section 5.3).

5.9 Conclusion

In this paper, we explored the potential of *automatically* detecting task switches and task types based on developers' computer interaction data. Based on two field studies that we conducted with a total of 25 professional software developers, we found that we are able to detect task switches and task types with high accuracy (81% and 61% respectively) and within a short time frame (average 1.6 minutes) from the actual task switch. We thereby examined a broad range of semantic and temporal features extracted from the computer interaction data and found that features based on user input data hold the highest predictive power. Our work extends previous work with an approach that uses a broader range of temporal and semantic features, by evaluating it in a field-study with professional developers, and by achieving higher accuracy than previous work while achieving a smaller delay in the predictions. The strong evidence on the potential to automatically predict task switches in the field opens up a wide range of application in real work settings, ranging from complementing existing *manual* task support, such as Mylyn, to automating time tracking tools, all the way to new tool support to leverage developers' workflows.

5.10 Acknowledgements

We thank our study participants for their participation. We also thank the anonymous reviewers and our editor for their valuable feedback.

6

Design Recommendations for Self-Monitoring in the Workplace: Studies in Software Development

***André N. Meyer**, Gail C. Murphy, Thomas Zimmermann, Thomas Fritz*
Published at the 2018 CSCW Conference on Computer-Supported Cooperative Work and Social Computing
Contribution: Study design and execution, tool development, data collection, data analysis, and paper writing

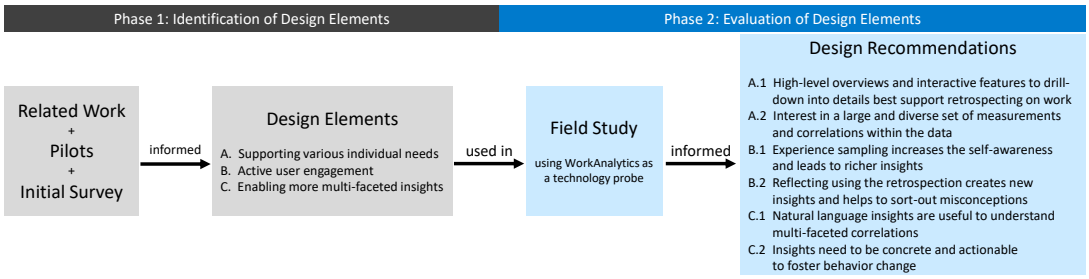
Abstract

One way to improve the productivity of knowledge workers is to increase their self-awareness about productivity at work through self-monitoring. Yet, little is known

about expectations of, the experience with, and the impact of self-monitoring in the workplace. To address this gap, we studied software developers, as one community of knowledge workers. We used an iterative, user-feedback-driven development approach (N=20) and a survey (N=413) to infer design elements for workplace self-monitoring, which we then implemented as a technology probe called *PersonalAnalytics*. We field-tested these design elements during a three-week study with software development professionals (N=43). Based on the results of the field study, we present design recommendations for self-monitoring in the workplace, such as using experience sampling to increase the awareness about work and to create richer insights, the need for a large variety of different metrics to retrospect about work, and that actionable insights, enriched with benchmarking data from co-workers, are likely needed to foster productive behavior change and improve collaboration at work. Our work can serve as a starting point for researchers and practitioners to build self-monitoring tools for the workplace.

6.1 Introduction

Figure 6.1: Summary of the Two-Phase Study Describing the Process.



The collective behavior of knowledge workers at their workplace impacts an organization's culture [Brown et al., 2014], success [Hofstede, 1994] and productivity [Mark et al., 2016b; Meyer et al., 2017a]. Since it is a common goal to foster productive behavior at work, researchers have investigated a variety of factors and their influence on knowledge workers' behavior and productivity, including the infrastructure and office environment [Brown et al., 2014; DeMarco

and Lister, 1985], the interruptions from co-workers [Chong and Siino, 2006; Czerwinski et al., 2004], and the teams' communication behaviors [Mark et al., 2016a; Meyer et al., 2014]. Yet, knowledge workers are often not aware of how their actions contribute to these factors and how they impact both their own productivity at work and the work of others [Perry et al., 1994b].

One way to improve knowledge workers' awareness of their own behavior and foster productive behavior is to provide them with the means to self-monitor and to reflect about their actions, for example through visualizations [Prochaska and Velicer, 1997]. This type of self-monitoring approach has been shown to foster behavior change in other areas of life, such as physical activity (*e.g.*, [Consolvo et al., 2008a; Fritz et al., 2014]), health (*e.g.*, [Bentley et al., 2013; Consolvo et al., 2008b]) and nutrition (*e.g.*, [Gasser et al., 2006]). Existing efforts to map the success of these self-monitoring approaches to the workplace have largely focused on tracking and visualizing data about computer use [Kim et al., 2016; RescueTime, 2019; Treude and Storey, 2010; Whittaker et al., 2016]. Although research has shown that self-monitoring at work can be valuable in increasing the awareness about a certain aspect of work, such as time spent in applications [Rooksby et al., 2016; Whittaker et al., 2016] or distracting activities [Kim et al., 2016], little is known about knowledge workers' expectations of and experience with these tools [Rooksby et al., 2016; Treude and Storey, 2010]. The lack of research about what knowledge workers' need from these tools may be one reason why many existing solutions have a low engagement and only short-term use overall [Collins et al., 2014; Kim et al., 2016]. Furthermore, most of these approaches did not consider collaborative aspects of work, such as instant messaging, email or meetings.

We address these gaps by aiming to better understand what information and features knowledge workers expect in workplace self-monitoring tools. To make our investigations tractable, we focus on one community of knowledge workers, software developers, before generalizing to a broader range of knowledge workers in the future. We study software developers due to their extensive use of computers to support both their individual and collaborative work, including the use of issue trackers for collaborative planning [Storey et al., 2014, 2017],

code review systems for shared feedback gathering [Bacchelli and Bird, 2013], and version control systems for co-editing artefacts [Vasilescu et al., 2016b]. Software developers are also an attractive target given the frequent interest of this community to continuously improve their work and productivity [Humphrey, 2000; Li et al., 2015]. Furthermore, software developers pursue a variety of different activities at work [Arciniegas-Mendez et al., 2017; Czerwinski et al., 2004; González and Mark, 2004] that vary considerably across work days and individuals [Meyer et al., 2017a]. For our investigations, this combination of diversity in activity, similarity in domain and extensive use of a computers yields an ideal combination for considering self-monitoring in the workplace.

To determine a set of design recommendations for building workplace self-monitoring tools, we followed a mixed-method approach, which is summarized in Figure 6.1. Phase 1 of our approach started with an investigation of software developers' expectations of and requirements for measures to self-monitor their work. A review of related work indicated barriers that have been identified towards the adoption of self-tracking technologies at the workplace, including not fully understanding users' needs [Li et al., 2010], not knowing in what measures users are interested in [Meyer et al., 2014; Rooksby et al., 2016; Treude et al., 2015], and not providing users with a holistic understanding of their work behavior [Ancker and Kaufman, 2007; Bartram, 2015; Galesic and Garcia-Retamero, 2011; Huang et al., 2016]. To overcome barriers associated with appropriate measures, we analyzed previous work on measures of software development productivity (*e.g.*, [Meyer et al., 2014]) and designed and developed a prototype, called $PersonalAnalytics_{pilot}$, that captures software development measures, allows software developers to self-monitor their work patterns and provides a retrospective view to a developer of their work day and work week.

We received feedback on the prototype through a pilot study with 20 participants and 5 iterations. Based on what we learned from the pilots, we conducted a study to learn about the design elements, including measures, needed in a self-monitoring tool for software development. We received input from 413 software development professionals for the survey. An analysis of the pilot and survey data indicated three design elements needed to build soft-monitoring

tools for a workplace: A) supporting various individual needs for data collection and representation, B) enabling active user engagement, and C) enabling more insights on the multi-faceted nature of work.

In phase 2, we then refined the prototype to accommodate these design elements and conducted a field study involving 43 professional software developers using the refined prototype for three weeks. The refined prototype, which we refer to as *PersonalAnalytics*, captures information from various individual aspects of software development work, including application use, documents accessed, development projects worked on, websites visited, as well as collaborative behaviors from attending meetings, and using email, instant messaging and code review tools. In addition, *PersonalAnalytics* prompts a user to reflect on their work periodically and to-self report their productivity based on their individual definition. To enable more multi-faceted insights, the captured data is visualized in a daily retrospection (see Figure 6.2), which provides a higher-level overview in a weekly summary, and allows users to relate various data with each other.

From the field study, we derived six design recommendations, summarized in Figure 6.1. For instance, we learned that a combination of self-reflection on productivity using self-reports, and observations made from studying the insights in the retrospection enhances participants' awareness about the time spent on various activities at work, about their collaboration with others, and about the fragmentation of their work. In this paper, we report on these six design recommendations and further requests made by participants for features to help them turn retrospective information into action. For instance, participants requested recommendation tools to help them better plan their work, improve their team-work and coordination with others, block out interruptions, and increase their productivity.

This paper provides the following main contributions:

- It demonstrates that self-monitoring at work can provide novel insights and can help to sort out misconceptions about work activities, but also highlights the need for information presented to be concrete and actionable, rather than simply descriptive.

- It demonstrates the value of brief and periodic self-reports to increase awareness of work and productivity for software developers.

- It presents a set of measurements specific to software development that professional software developers report to provide the most value to increase awareness of their work, ranging from the time spent doing code reviews to the number of emails received in a work day.

This paper is structured as follows: We first discuss related work before we present how we identified design elements for self-monitoring in the workplace, and how we incorporated and evaluated them using *PersonalAnalytics* as a technology probe. Subsequently, the findings and distilled design recommendations are presented. Finally, we discuss our findings with respect to long-term user engagement, potential impact on individuals and the collaboration with their teams, and the generalizability of our results.

6.2 Related Work

Previous work on approaches for self-monitoring various aspects of life and work is discussed in Section 1.6 of the synopsis to avoid repetitions. Barriers towards the adoption of these self-tracking technologies that are specific to this publication are presented below.

6.2.1 Designing and Evaluating Self-Monitoring Tools for Work

Only few of the workplace self-monitoring tools we presented in Chapter 1.6.6 have been evaluated (*e.g.*, [Huang et al., 2016; Kim et al., 2016; Rooksby et al., 2016; Whittaker et al., 2016]), limiting our knowledge of the overall value of these tools to users, particularly limiting our knowledge of which information is of value to users and if the approaches can affect the behaviour of users. As described by Klasnja et al. [2011], it is often feasible to evaluate the efficacy of a self-monitoring tool in a qualitative way to identify serious design issues early, while still seeing trends in how behaviour might change in the long-term. In this

paper, we follow this recommendation, focusing on facilitating the reasoning and reflection process of a knowledge worker by increasing self-awareness about the monitored aspect of work [Huang et al., 2016; Kim et al., 2016; Prochaska and Velicer, 1997]. We leave an assessment of whether the design recommendations we provide can be embodied in a tool to change user behaviour to future work. To provide a starting point for building self-monitoring tools targeting software developers at work and evaluate their potential impact on behaviors at work, we conducted a three-week user study to investigate the efficacy of the design elements that we identified from related work, five pilots, and a survey, using *WorkAnalytics* as a technology probe. To our knowledge, this is also the first approach that focuses to raise developers' awareness about their collaborative activities, such as gaining insights about emailing, meeting, and code reviewing.

Previous research has also discovered that users rarely engage with the captured data, resulting in a low awareness and reducing chances for a positive behavior change when using a self-monitoring tool [Collins et al., 2014; Huang et al., 2016; Kim et al., 2016]. We compiled and categorized a list of barriers related work has identified towards the adoption of self-monitoring technologies at the workplace:

Not understanding user needs. Research has shown that knowledge workers' needs for monitoring their computer use vary and that little is actually known about the measures they are interested in [Li et al., 2011; Meyer et al., 2014; Rooksby et al., 2016; Treude et al., 2015]. Users sometimes also have too little time for a proper reflection of the data, or an insufficient motivation to use the tool, which is likely one reason they often stop using it after some time [Li et al., 2010]. This emphasizes the importance of understanding users' needs and expectations about how self-monitoring tools should work and what measures they should track, to increase the chance people are trying such a tool and using it over extended periods.

Lack of data context. Most tools we found miss the opportunity to provide the user with a more holistic understanding and context of the multi-faceted nature of work, as they only collect data about a single aspect, *e.g.*, the programs used on the computer [Choe et al., 2014; Epstein et al., 2016a]. This makes it

difficult for users to find correlations between data sets and, thus, limits the insights they can get. Behavior change cannot be modelled based on just a few variables, as the broader context of the situation is necessary to better understand the various aspects influencing work behavior and productivity [Bartram, 2015; Huang et al., 2016]. To overcome this, Huang et al. [2016] propose to integrate these self-monitoring approaches into existing processes or tools and place them into an already existing and well-known context which makes it easier for users to engage in an ongoing tool use. Choe et al. [2014] further suggest to track many things when users first start a self-monitoring initiative, and then let them decide which measures are necessary for their context to reflect and improve their behavior.

Difficulties in interpreting the data. Choe et al. [2014] and Huang et al. [2016] argue how difficulties in making sense of, organizing or interpreting the data result in a lower adoption of self-monitoring approaches, as users will stop using them. For example, Galesic and Garcia-Retamero [2011] found that more than 40% of Americans and Germans lack the ability to understand simple graphs, such as bar or pie charts, which could be a problem for self-monitoring tools as they often visualize the data. To overcome this issue, Bentley et al. [2013] propose to provide insights from statistically significant correlations between different data types in natural language, which helped participants in the study to better understand the data. Another problem to efficiently interpret data in personal informatics systems is information overload, as described by Jones and Kelly [2017]. They found that users generally have a higher interest in multi-faceted correlations (correlations between two distinct data categories), rather than uni-faceted correlations, that reveal "surprising" and "useful" information. Hence, this could help to reduce information overload and provide more relevant insights to users.

Privacy Concerns. Another potential pitfall of self-monitoring tools is data privacy, as many users are afraid the data might have a negative influence on their life, such as fearing their managers may know how well they sleep, or that their insurance agency can track their activity. Most privacy concerns can be reduced by letting users decide what and how they want to share their data, by

obfuscating sensitive data when it is being shared, by abstracting visualizations, and by letting users opt-out of applications when they think the gained benefits do not outweigh the privacy risks [Begole et al., 2002; Mathur et al., 2015].

Besides learning more about software developers' expectations of and experience with a self-monitoring tool for work and productivity, we used our iterative, feedback-driven development process and a survey to investigate how these barriers could be tackled. Based on the findings, we incorporated the identified design elements into our self-monitoring approach *PersonalAnalytics* and then used it to evaluate how the design elements affect developers' awareness on work and productivity. Subsequently, we distilled design recommendations for building self-monitoring tools for developers' work.

6.3 Phase 1 Method: Identifying Design Elements

To identify design elements for building personalized awareness tools for self-monitoring software developers' work, we defined the following research question:

> **RQ1:** What information do software developers expect and need to be aware of and how should this information be presented?

To answer this research question, we first reviewed literature of design practices applied in existing self-monitoring tools and of measures that software developers are interested in. We also studied the barriers related work has identified towards the adoption of self-tracking technologies at the workplace, as described in the previous section. Based on our review, we defined design elements and incorporated them into our own self-monitoring prototype for work, called $PersonalAnalytics_{pilot}$. We then studied software developers' use of and experience with $PersonalAnalytics_{pilot}$ at work, and refined the design elements and tool based on feedback we received through five pilots and a survey.

In what follows, we describe the goals, method and participants of this first phase. Table 6.1 shows an overview of the pilots and survey that we conducted and

Table 6.1: Overview of the Two-Phase Study Describing the Method, Participants, their Employer and Study-Durations.

Phase 1: Identification of Design Elements for Self-Monitoring at Work (iterative, feedback-driven development of WorkAnalytics)					
Method	# Partic.	Company ID	# Developers	Location	Duration
Pilots	**20**				**2-4 work weeks**
Pilot 1	6	A	ca. 3000	Canada	2 work weeks
Pilot 2	2	B	ca. 150	Canada	2 work weeks
Pilot 3	3	C	4	Switzerland	2 work weeks
Pilot 4	5	D	ca. 50000	USA	4 work weeks
Pilot 5	4	A	ca. 3000	Canada	3 work weeks
Initial Survey	**413**	**D**	**ca. 50000**	**USA**	**sent out 1600 invitations**

Phase 2: Evaluation of the Design Elements for Self-Monitoring at Work (using WorkAnalytics as a technology probe)					
Method	# Partic.	Company ID	# Developers	Location	Duration/Timing
Field Study	**43**	**D**	**ca. 50000**	**USA**	**3 work weeks**
Email Feedback	34				arbitrarily during the study
Intermed. Feedback Survey	26				after the first week
Data Upload	33				at the end of the study
Final Survey	32				following the data upload

situates them within the whole study procedure. The supplementary material [1] contains a list of questions for all surveys and interviews that we conducted as well as screenshots of how $PersonalAnalytics_{pilot}$ looked like at various stages until the final version.

6.3.1 Pilots

To examine the features and measurements software developers are interested in and engage with for self-monitoring their work from using them in practice, rather than from doing this hypothetically through an interview or survey, we conducted a set of pilots. Our method has strong similarities to the *Design Based Research* process, where the focus is an iterative analysis, design and implementation, based on a collaboration between practitioners and researchers in a real-world setting that leads to design principles in the educational sector [Brown, 1992]. First, we implemented a self-monitoring prototype, $PersonalAnalytics_{pilot}$, incorporating visualizations of work-related measures that we identified to be of

[1] `https://doi.org/10.5281/zenodo.884051`

interest to software developers in previous research from running a survey with 379 participants [Meyer et al., 2014]. We then conducted a total of five pilots at four companies (see Phase 1 in Table 6.1 for more details). For each pilot, we had a small set of software developers use $PersonalAnalytics_{pilot}$ *in situ*, gather their feedback, and use it to refine and improve the prototype before running the next pilot. Each pilot study ran between 2-4 work weeks. To gather feedback, we conducted interviews with each participant at the end of the pilot period. These interviews were semi-structured, lasted approximately 15 minutes, and focused on what participants would like to change in the application and what they learnt from the retrospection. To find and address problems early during the development, we also conducted daily 5 minute interviews with each participant during the first three pilots. In these short interviews, we gathered feedback on problems as well as changes they would like to be made to the application, and feedback on the visualizations, their representativeness of participants' work and their accuracy. Throughout this phase, we rolled out application updates with bug-fixes, updated visualizations and new features every few days. We prioritized user requests based on the feasibility of implementation and the amount of requests by participants. After 5 pilots we decided to stop since we did not gather any more new feedback and the application was running stable.

Participants. For the pilots, we used personal contacts and ended up with a total of 20 professional software developers, 1 female and 19 male, from four different companies of varying size and domains (Table 6.1). 30% reported their role to be a team lead and 70% an individual contributor—an individual who does not manage other employees. Participants had an average of 14.2 years ($\pm$9.6, ranging from 0.5 to 40) of professional software development experience.

6.3.2 Initial Survey

Following the pilot studies, we conducted a survey 1) to examine whether the measures and features that developers are interested in using for self-monitoring within the target company (company D) overlap with what we had implemented, 2) to learn how the $PersonalAnalytics_{pilot}$ needed to be adapted to fit into the target company's existing technology set-up and infrastructure, as well as 3) to

generate interest in participating in our field study. In the survey, we asked software developers about their expectations and the measurements that they would be interested in for self-monitoring their work. We advertised the survey at company D, sending invitation emails to 1600 professional software developers. To incentivize participation, we held a raffle for two 50 US$ Amazon gift certificates. The initial survey questions can be found in the supplementary material. To analyze the survey, we used methods based on Grounded Theory [Strauss and Corbin, 1998] to analyze the textual data that we collected. This included Open Coding to summarize and label the responses, Axial Coding to identify relationships among the codes, and Selective Coding to factor out the overall concepts, related to what measurements and features participants expect and how their work environment looks like.

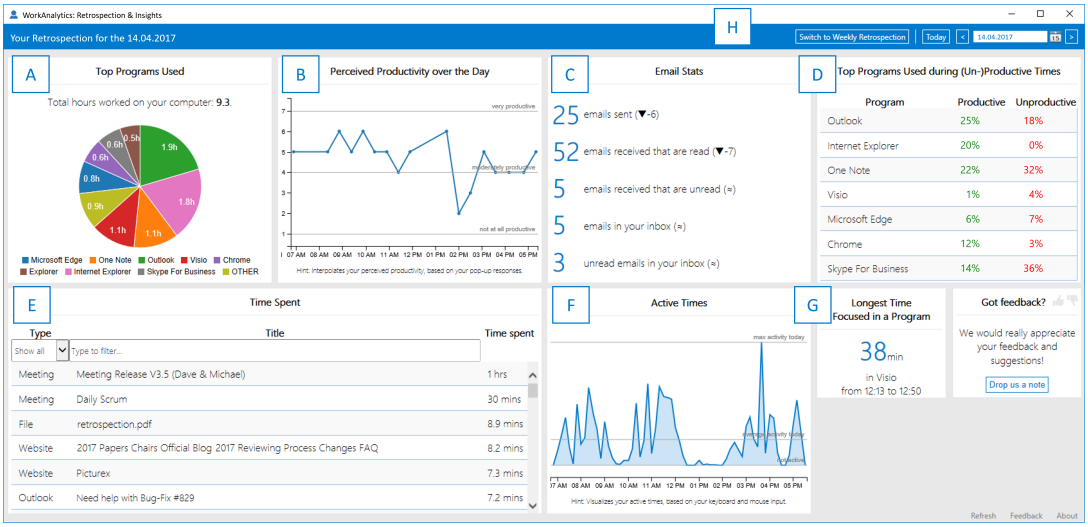
Participants. From the 1600 invitation emails, we received responses from 413 software developers (response rate: 25.8%), 11% female, 89% male. 91.5% of the participants reported their role to be individual contributor, 6.5% team lead, 1 manager (0.2%), and 1.8% stated they are neither. Participants had an average of 9.6 years (±7.5, ranging from 0.3 to 36) of professional software development experience.

6.4 Phase 1 Results: Identified Design Elements

To answer our first research question (**RQ1**), we analyzed related work, investigated developers' experience with pilots of $PersonalAnalytics_{pilot}$ and analyzed the initial survey. The analysis showed that a design for a work self-monitoring approach should: A) support various individual needs, B) foster active user engagement, and C) provide multi-faceted insights into work. We incorporated these three design elements into a technology probe, *PersonalAnalytics*.

PersonalAnalytics was built with Microsoft's Dot.Net framework in C# and can be used on the Windows 7, 8 and 10 operating system. We created *PersonalAnalytics* from the ground up and did not reuse an existing, similar application, such as RescueTime [2019], as we wanted to freely extend and modify all features and measurements according to our participants' feedback. A

Figure 6.2: Screenshot of the Daily Retrospection in *PersonalAnalytics*.



screenshot of the main view of the application, the retrospection, is shown in Figure 6.2. We open-sourced *PersonalAnalytics*, opening it up to contributions on GitHub [2].

6.4.1 A: Supporting Various Individual Needs

Measurement Needs. The analysis of our initial survey showed that participants are generally interested in a large number of different measures when it comes to the self-monitoring of work. We asked survey participants to rate their interest in a list of 30 work related measures on a five point Likert-scale from 'extremely interesting' to 'not at all interesting'. We chose these measures based on our findings from the pilot phase, on what we were capable to track, and on related work. The list includes measures on time spent in programs, meetings, and specific activities, the amount of code written, commits done, code reviews completed, emails sent and received, and the amount of interruptions experienced and focus at work. Each measure had at least 20% and up to 74% of the participants that rated it as very or extremely interesting. At the same time the combination of measures that each participant was interested in varied greatly

[2] `https://github.com/sealuzh/PersonalAnalytics`

across participants. For instance, only 6 of the 30 measures were rated as very or extremely interesting by 60% or more, and 52% of participants were interested in nearly all measures while 25% only wanted very few measures for self-monitoring at work. Overall, the greatly varying interest and the interest in a large number of measures for self-monitoring supports earlier findings by Meyer et al. [2014] in the work domain and Choe et al. [2014] in the activity and health domain. The complete list of the 30 work related measures, including participants' ratings about their interest in the measures, can be found in the supplementary material.

To support these individually varying interests in work measures, we included a wide variety of measures in our application and allowed users to individually select the measures that were tracked and visualized. To capture the relevant data for these measures, *PersonalAnalytics* features multiple data trackers: the *Programs Used* tracker that logs the currently active process and window titles every time the user switches between programs or logs 'idle' in case there was no user input for more than 2 minutes; the *User Input* tracker, to collect mouse clicks, movements, scrolling and keystrokes (no key-logging, only time-stamp of any pressed key); and, the *Meetings and Email* trackers, to collect data on calendar meetings and emails received, sent and read, using the MicrosoftGraphApi [2019] of the Office 365 Suite.

The initial version only included the Programs Used tracker, similar to RescueTime [2019]. The Programs Used tracker allows the extraction of a multitude of measurements participants wished, including the time spent in specific programs and activities, such as development related activities (*e.g.*, coding, testing, debugging, version control, and development projects worked on) and researching the web, as well as specific code files and documents worked on and websites visited. After the first two pilots, the User Input tracker was added, since 3 of the first 8 participants were interested in knowing when they were producing (*e.g.*, typing on the keyboard) and consuming (*e.g.*, scrolling through text with the mouse) data. Running the initial survey highlighted participants' interest in knowing more concrete details about their collaborative activities, such as planned and unplanned meetings (41%), reading and writing emails (44%), and doing code reviews (47%), which is the reason they were added to

the final version of *PersonalAnalytics* before running the field study.

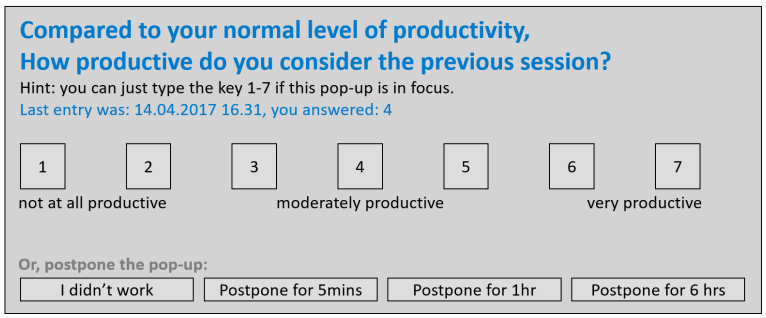
Privacy Needs. A re-occurring theme during the pilots and initial survey was participants' need to keep sensitive workplace data private. Participants feared that sharing data with their managers or team members could have severe consequences on their employment or increase pressure at work. To account for privacy needs at work, *PersonalAnalytics* stores all logged data only locally on the user's machine in a local database, rather than having a centralized collection on a server. This enables users to remain in control of the captured data. To further support the individual needs, the application provides actions to enable and disable data trackers manually, pause the data collection and access (and alter) the raw dataset, which was done by two participants during the field study.

6.4.2 B: Active User Engagement

To be able to generate deeper insights on a user's work and productivity and encourage users to actively reflect upon their work periodically, we decided to include a self-reporting component. Several participants of our initial survey stated interest in self-reporting some data about work that cannot be tracked automatically, in particular more high-level measures on productivity. Furthermore, related work found that users rarely engage by themselves with data captured in a self-monitoring tool, which reduces awareness and chances of positive change [Collins et al., 2014; Huang et al., 2016; Kim et al., 2016]. To address this point, we added a pop-up to our application that appeared periodically, by default once per hour[3], and prompted users to self-report their perceived productivity, the tasks they worked on, the difficulty of these tasks and a few other measures. During the first two pilots of our iterative development phase, we found that while the self-reporting might be valuable, it took participants several minutes to answer, and 45% of our participants reported it to be too intrusive, interrupting their work, and decreasing their productivity. As a result,

[3]This interval was chosen as a way to balance intrusiveness. While the first two pilots had an interval of 90 minutes which made it harder for participants to remember what exactly happened in that period, most participants preferred to reflect on their productivity once an hour.

Figure 6.3: Screenshot of the Self-Reporting Pop-Up to Collect Perceived Productivity Data and Engage Users.



many participants regularly postponed the pop-up or disabled it, which then resulted in less meaningful observations to be presented in the visualization and a smaller satisfaction by participants.

To minimize intrusiveness, yet still encourage periodic self-reflection, we reduced the number of questions in the pop-up to a single question that asks participants to rate their perceived productivity on a 7-point Likert-scale (1: not at all productive, 7: very productive) once per hour. Participants were able to answer the question with a single click or keystroke. See Figure 6.3 for a screenshot of the pop-up. In case the pop-up appeared at an inopportune moment, participants were able to postpone it for a few minutes, an hour or a whole work day. To further adapt the self-reports to individual preferences, each participant was able to alter the interval at which pop-ups appeared or disable/enable it.

6.4.3 C: Enabling More Multi-Faceted Insights

Related work found that self-monitoring tools often fail to provide sufficient contextual information and a more holistic picture of the monitored behavior that also allows the user to relate the data [Bartram, 2015; Choe et al., 2014; Huang et al., 2016]. Similarly, 35% of pilot study participants asked for weekly summaries to get a more complete picture of the data and a way to compare and relate different work days or weeks with each other. In the initial survey, 41% of the participants wished for a visualization to drill down into the data and learn

where exactly they spend their time.

To address this requirement of enabling a more complete picture of the data in our application, we focused on three aspects: providing sufficient contextual information, allowing to get a higher-level overview, and providing ways to relate various data with each other. To provide *sufficient contextual information*, we added several visualizations to the daily retrospection that illustrate how the time of a work day was spent:

- **Top Programs Used:** Pie chart displaying the distribution of time spent in the most used programs of the day (Figure 6.2A).

- **Perceived Productivity:** Time line illustrating the user's self-reported productivity over the course of the day (Figure 6.2B).

- **Email Stats:** Table summarizing email related data, such as number of emails sent & received in a work day (Figure 6.2C).

- **Programs & Productivity:** Table depicting the seven most used programs during the day and the amount of time the user self-reported feeling productive versus unproductive while using them (Figure 6.2D).

- **Time Spent:** Table showing a detailed break-down of how much time was spent on each information artefact during the work day, including websites visited, files worked on, emails sent/read, meetings in the calendar, as well as code projects and code reviews worked on (Figure 6.2E).

- **Active Times:** Line chart visualizing the user's keyboard and mouse input over the course of the day. We aggregated the input data by assigning heuristic weights to each input stream that we determined based on our own experience and trials in pilots, *e.g.*, one mouse click has approximately as much weight assigned as three key strokes (Figure 6.2F).

- **Longest Time Focused:** Minutes that a user spent the longest inside any application without switching (Figure 6.2G).

For a *higher-level overview*, we added a weekly summary of the data, which shows how often which programs were used on each day of the week, the average self-reported productivity per day, and the productive versus unproductive time spent on the 7 most used programs during the week (same as Figure 6.2E). The supplementary material contains a screenshot of the weekly retrospection.

Finally, to *ease the correlation of data*, as desired by 19% of the participants in the initial survey, we implemented a feature that allows users to pick days or weeks (Figure 6.2H) and compares them with each other side-by-side and we provide a view that correlates the most used programs during a day with productivity (Figure 6.2D). In addition to these features, we automatically generated personalized insights. Personalized insights are automatically generated aggregations and correlations within the captured data and presented in natural language. These personalized insights are similar to the correlation and presentation of data that Bentley et al. [2013] have shown to increase users' understanding of complex connections in the area of health-monitoring and well-being. To create the personalized insights, we first created a matrix where we correlated each measure with itself (i.e. average per day), with the time of the day (i.e. morning/afternoon), and with the productivity self-reports. To avoid information overload, we just selected insights that might be interesting to users by discarding simple insights from the matrix that were already easily perceptible in the retrospection (*e.g.*, the number of emails sent per day or user input over the day) and removed one insight that we could not produce due to the format of the collected data (number of emails sent/received over the day). For each pair, we created one or more sentences that correlate the items with each other. For example, from the pair 'self-reported productivity' and 'time of day', we created the sentence: "You feel more productive in the [morning/afternoon]" (insight 14). Three of these personalized insights address the participants' focus, which is an abstracted measure for the time spent in a single program before switching to another program. Participants were aware of the definition of focus, as one of the visualizations in the daily retrospection used the same abstraction and included a definition (Figure 6.2G). We created these personalized insights individually for each user and filtered the ones that were not feasible, *e.g.*, due to

participants disabling certain data trackers. Since we wanted to ensure to collect sufficient data before generating these personalized insights and also ensure that they are reasonable, we only included them in the final survey, after users shared their data logs with us. Table 6.3 presents a list of the 15 personal insights that resulted from this process. The matrix we created to select these insights is available and discussed in the supplementary material. Future versions of *PersonalAnalytics* will include the automatic generation of such personalized insights.

6.5 Phase 2 Method: Evaluating Design Elements

To evaluate the design elements, and learn how software developers are using and appreciating the identified features and measurements in practice, we formulated a second research question:

> **RQ2:** How do software developers use the measurements and features based on the identified design elements during their work and what is their impact?

To answer the research question, we conducted a field study with *PersonalAnalytics* as a technology probe that implements the previously discussed design elements.

6.5.1 Participants

We recruited participants for this study by contacting the 160 software developers at company D that took our initial survey and indicated their interest in participating. 33 of the 43 participants that signed the consent form were recruited through this follow-up email, and 10 participants were recruited through recommendations from other participants. The only requirements for participating in the study were to be a software developer and to be using a work machine with the Windows operating system. Participants were given two 10 US$ meal cards at the end of the study for compensating their efforts and were promised personalized insights into their work and productivity. All 43 participants are professional software developers working in the same large software company

(company D in the pilots), three of them were female and 40 male. The roles, team sizes and projects varied across the participants. 96.7% stated their role to be an individual contributor and 3.3% team lead. Participants had an average of 9.8 years (±6.6, ranging from 0.5 to 30) of professional software development experience. To avoid privacy concerns, we identified participants with a subject id and therefore could not link their responses between the different feedback surveys, emails, and collected data from *PersonalAnalytics*. To get feedback on the usefulness of the different design elements from different perspectives, we picked participants with and without previous experience with other self-monitoring tools, such as Fitbit [2019] or RescueTime [2019].

6.5.2 Procedure

We designed this field study to last three work weeks. At the beginning of the period, we provided participants with detailed information on the study procedure, the data we were going to collect, and the features of *PersonalAnalytics*. We then asked participants to install the application on their work machine, continue their regular work day and answer the periodic self-reports when they appeared, by default every 60 minutes. We asked them to contact us via email at any point in time in case they run into an issue, had questions, or suggestions, which 34 participants did once or more. At any point throughout the study, participants were able to change the time period or disable the pop-up completely. Participants could also enable or disable any trackers that logged data for presentation in the retrospection. After the first week, we sent out a short, intermediate feedback survey to collect early feedback on the usefulness, suggestions for improvement, and participants' engagement with *PersonalAnalytics*. 26 participants responded. The timing was chosen to make sure participants had used the application for at least 3 to 5 work days, and the tool had captured enough data to show visualizations from various work days.

Shortly before the end of the three work weeks of the study, we asked participants to share the data that *PersonalAnalytics* logged on their machine—the reported productivity ratings and the computer interactions—if they were willing to. We also gave each participant the opportunity to obfuscate any

sensitive or private information that was logged, such as window titles or meeting subjects, before uploading the data to our secured server. Of the 43 participants, 33 participants shared their data with us, and three of them obfuscated the data before the upload. Due to the sensitivity of the collected data, we did not try to convince participants to share the data and just mentioned the additional insights they would receive when sharing it. We then used the data to automatically generate aggregations and correlations within an individual participant's data, which we will call personalized insights in the following. At the end of the study period, we asked participants to fill out a final survey, independently of whether they uploaded the data or not. The survey contained questions on feature usage and usefulness, possible improvements, potential new measures, and perceived changes in awareness about work and behavior. For participants that shared the collected data with us, the survey also presented the personalized insights, automatically generated for each participant, and questions about them. 32 of the 43 participants participated in the final survey, including 5 that had not previously shared their computer interaction data. The questions from the intermediate survey and final survey can be found in the supplementary material.

6.5.3 Data Collection and Analysis

Throughout the field study, we collected qualitative and quantitative data from participants. In particular, the responses to the intermediate feedback survey, final survey, feedback received via email, and the data that *PersonalAnalytics* collected. Similar to our approach in the initial survey, we used methods common in Grounded Theory. In this case, the Axial Coding step was also used to identify higher level themes after Open Coding each feedback item separately. Besides creating personalized insights from the collected computer interaction data, we used it to analyze participants' engagement with the retrospection and the answering of the experience sampling productivity pop-up. The computer interaction data span over a period of between 9 and up to 18 work days (mean=13.5, $\pm$2.6). The findings from analyzing the quantitative and qualitative data of our participants are discussed and distilled into design recommendations in the next section.

6.6 Phase 2 Results: Design Recommendations Based on Evaluating Design Elements

To answer the second research question (**RQ2**), we focus our analysis of the data collected about the use of *PersonalAnalytics*. For each part, we first present the findings before summarizing the design recommendations that we inferred from interpreting the results. The design recommendations are mapped to one of the three design elements (A to C) and are presented in blue boxes to distinguish them from the findings.

6.6.1 Different Granularity of Visualizations

Most participants (70.4%) agreed that the collected data and measures were interesting and relevant to them. Participants valued that the retrospection allowed them to get a high-level overview of the data and also let them drill down into more detail:

> "*Sift through all this information and quickly find what's critical and be able to determine what is furthering one's goals and what [is] not (i.e. is a distraction).*" - F19

Participants used, for instance, the pie chart on the programs executed (Figure 6.2A) and the active times timeline (Figure 6.2F) to get an aggregated overview of the past work day, in particular which activities most time was spent on and the most active and inactive times during the day, respectively. When they wanted to further investigate their day and find out more specific details, participants appreciated the availability of other visualizations:

> "*I like that [WorkAnalytics] captures who I am talking with in Skype or Google Hangouts [. . .]. I like the integration of Outlook in more detail.*" - F42

Several participants (F13, F17, F18) reported having used the time spent table (Figure 6.2E) regularly to gain deeper insights on with whom they communicate—through email, instant messaging and meetings—and on which artefacts they spent time—document, website, code file, or email.

Design Recommendation A.1: For self-monitoring at work users are interested in a quick as well as deep retrospection on their work that are best supported through high-level overviews with interactive features to drill-down into details.

6.6.2 Interest in Diverse Set of Measurements

Participants had varying interests in the positive, negative or neutral framing of the data. For instance, while some participants (F19, F25) wanted to learn about what went well, such as the tasks they completed and how much they helped their co-workers, others were more interested in understanding what went wrong:

> "*[...] focus more on things that prevent someone from being able to add business value, rather than arbitrary metrics like commit count, bug count, task completion, etc. [...] I would prefer [the application] to track things that I felt got in the way of being productive.*" - F17

This framing effect in self-monitoring tools has recently been explored by Kim et al. [2016], where they found out that only participants with a negative framing condition improved their productivity, while positive framing had little to no impact.

Most participants (69%) wanted *WorkAnalytics* to collect even more data on other aspects of their work to further personalize and better fit the retrospection to their individual needs. For instance, they wanted more detailed insights into collaborative and communicative behaviors by integrating data from and sharing data with other team members (6%) and generating insights into the time spent on technical discussions or helping co-workers (6%). Participants were further interested in collecting data from other work devices (13%), capturing even more coding related data (6%), such as tests and commits, or more high-level measures, such as interruptions or progress on tasks (9%). 80% of the participants were also interested in biometric data, such as heart rate or stress levels, 70% were interested in physical activity data, such as sleep or exercise, and 50% were

interested in location based data, such as commute times or visited venues; all in combination with the already collected work data. Similarly, roughly one third of the participants suggested to extend the daily and weekly retrospection, by adding additional visualizations and finer-grained aggregations, to better support them in making observations based on correlations and combinations of several measurements:

> "*[The] active times graph would be neat on the weekly retrospection so that I could get a sense of my most active time of the day without having to navigate through each day.*" - F43

These very diverse requests for extending *WorkAnalytics* with further measures and visualizations emphasize the need for personalizing the experience, to increase satisfaction and engagement.

Design Recommendation A.2: For self-monitoring one's work, users are interested in a large and diverse set of data, even from outside of work, as well as in correlations within the data.

6.6.3 Increasing Self-Awareness with Experience Sampling

Participants actively engaged in the brief, hourly self-reports on productivity when they were working on their computer. Over the course of the study, participants self-reported their productivity regularly, on average 6.6 times a day (± 3.8, min = 1, max = 23) and it usually took them just a couple of seconds, without actually interrupting their work. Two (6%) participants even increased the frequency to answer the pop-up every 30 minutes, while 3 (9%) of the 33 participants, from whom we received data, disabled the self-reports. This shows that the experience sampling method we applied was not considered as too intrusive for most participants.

Being asked in the final survey about the value of and experience with self-reporting their productivity, 59.2% of the participants agreed or strongly agreed that the brief self-reports increased their awareness on productivity and work (see Table 6.2 for more detail). The self-reports helped participants to realize how they have spent their past hour at work and how much progress they have made on the current task:

> "*It makes me more conscious about where I spent my time and how productive I am.*" - F08

Some participants used the pop-up to briefly reflect on whether they have used their time efficiently or not, and if they should consider changing something:

> "*The hourly interrupt helps to do a quick triage of whether you are stuck with some task/problem and should consider asking for help or taking a different approach.*" - F11

The fact that *PersonalAnalytics* does not automatically measure productivity, but rather lets users self-report their perceptions, was further valued by participants as some do not think an automated measure can accurately capture an individual's productivity, similar to what was previously found by Meyer et al. [2017a]:

> "*One thing I like about [WorkAnalytics] a lot is that it lets me judge if my time was productive or not. So just because I was in a browser or VisualStudio doesn't necessarily mean I was being productive or not.*" - F42

> "*I am much more honest about my productivity levels when I have to self-report, [rather] than if the software simply [...] decided whether or not I was productive.*" - F15

These findings suggest that using experience sampling is a feasible method to manually collect data as long as users have a benefit from their self-reporting.

Design Recommendation B.1: Experience sampling in the form of brief and periodic self-reports are valuable to users as they increase the awareness of their work and productivity, and lead to richer insights.

6.6.4 Increasing Self-Awareness with a Retrospection

Participants frequently accessed the daily retrospection, yet the patterns of self-monitoring varied greatly across participants. On average, participants opened the retrospection 2.5 times per day (±3.5, min=0, max=24) for a total of 0.85 minutes (±2.95, min=0, max=42.9), but both aspects varied a lot across

participants as the standard deviation (±) and the minimum and maximum show. All participants opened the retrospection more often in the afternoon (mean=1.9) than in the morning (mean=0.6). Yet, 34% of participants opened the application less than 5 times over the whole study period, while 28% used the retrospection at least once a day. Also, while 31% of participants mostly focused on the current day, the other 69% looked and compared multiple work days. Many participants also looked at the weekly retrospection, but access to this one was less often than to the daily one.

While these results show that most participants were actively reflecting about their work using the retrospection, we also received feedback from 2 participants (6%) that they sometimes forgot the retrospection was available:

> "*I forgot I could even look at the retrospection! A new pop-up, maybe Friday afternoon or Monday morning prompting me to review the week's data would be really nice.*" - F14

Overall, the retrospection increased the awareness of the participating software developers and provided valuable and novel insights that they were not aware of before. Overall, participants commented on the retrospection providing novel insights on a variety of topics, such as how they spend their time at work collaborating or making progress on tasks, their productivity over the course of the day, or the fragmentation and context switches at work:

> "*Context switches are not the same as program switches, and I do *lots* of program switches. I still do a lot more context switches than I thought, but it doesn't hurt my perceived productivity.*" - F36

> "*[The] tool is awesome! It [...] helped confirm some impression I had about my work and provided some surprising and very valuable insights I wasn't aware of. I am apparently spending most of my time in Outlook.*" - F42

Reflecting about the time spent at work further helped participants to sort out misconceptions they had about their work:

> "*I did not realize I am as productive in the afternoons. I always thought my mornings were more productive but looks like I just think that because I spend more time on email.*" - F14

Table 6.2: Survey Responses on Awareness Change.

	Strongly agree	Agree	Neutral	Disagree	Strongly disagree	N/A
The collected and visualized data is relevant to me.	18.5%	51.9%	22.2%	7.4%	0.0%	0.0%
I learned something about my own work and perceived productivity by looking at the retrospection and reflecting.	29.6%	29.6%	25.9%	11.1%	0.0%	3.7%
Answering the perceived productivity pop-up questions increased my awareness about my work and perceived productivity.	18.5%	40.7%	25.9%	7.4%	7.4%	0.0%
Installing and running the tool raised my awareness about my work and perceived productivity.	22.2%	59.3%	11.1%	3.7%	3.7%	0.0%
I used the daily retrospection to reflect about my past work day.	11.1%	37.0%	11.1%	29.6%	7.4%	3.7%
I used the weekly retrospection to reflect about my past work week.	11.5%	30.8%	23.1%	23.1%	7.7%	3.8%
The retrospection helps me to learn how I spend my time.	29.6%	55.6%	0.0%	11.1%	0.0%	3.7%
The retrospection helps me to learn more about my perceived productive times.	25.9%	33.3%	25.9%	7.4%	3.7%	3.7%
I now know more about why and when I feel I am productive or unproductive.	22.2%	40.7%	14.8%	18.5%	3.7%	0.0%
I tried to change some of my habits or patterns based on what I learned from reflecting about my work.	14.8%	25.9%	11.1%	40.7%	3.7%	3.7%

The survey responses that are presented in Table 6.2 and are based on a 5-point Likert-scale (5: strongly agree, 1: strongly disagree) further support these findings. 81.5% of all survey participants reported that installing and running the application increased their awareness, and 59.2% agreed or strongly agreed that they learnt something about their work and productivity, while only 11.1% did not. The responses also show that the retrospection helped participants in particular to learn how they spend their time (85.2% agreed or strongly agreed) and about productive and unproductive times (62.9%).

Design Recommendation B.2: Reflecting about work using a retrospective view provides novel and valuable insights and helps to sort out misconceptions about activities pursued at work.

6.6.5 Personalized Insights

The personalized insights that we presented to 27 of the 32 participants in the final survey are based on the same measurements as the ones that are visualized in the retrospection. These insights were created based on correlations and aggregations within the collected data and presented as natural language sentences. The specific insights are presented in Table 6.3 and details on their creation can be found in Section 6.4.3. To learn more about the value of the visualizations and the natural language insights, we asked participants to rate the novelty of each personalized insight. Participants' responses were mixed with respect to the novelty of the automatically generated personalized insights that presented correlations and aggregates within the data in natural language. When rated on a scale from 'extremely novel' to 'not novel at all', only 5 of the 15 personalized insights (see personalized insights in Table 6.3 marked with an asterisk '*') were rated as 'very novel' or 'extremely novel' by more than half of the participants. This means that participants gained knowledge about most insights either before or during the study. The five insights that were rated as 'very novel' or 'extremely novel' by more than half of the participants are all correlations between two distinct data categories, so called multi-faceted correlations [Jones and Kelly, 2017], rather than simple aggregates, called uni-faceted correlations, which are easier to understand from simple visualizations [Bentley et al., 2013; Galesic and Garcia-Retamero, 2011]. One participant also suggested to integrate these novel personalized insights into the retrospection since it was easier to draw connections between two distinct data categories using natural-language statements, similar to what Bentley et al. [2013] found. Research by Jones and Kelly [2017] has shown that multi-faceted correlations presented by self-monitoring tools are of higher interest to users than uni-faceted correlations. Paired with our findings above, this suggests to use visualizations for presenting uni-faceted correlations and to present more complex multi-faceted correlations using natural language sentences. Future work could further investigate the effectiveness of these personalized insights and their impact on behavior at work.

Table 6.3: Participants' Ratings on the Novelty and Potential for Behavior Change of Personalized Insights.

	Novelty				Behavior Change		
	extremely	very	somewhat	not	yes	no	dk
1. The program you spend most time is X, followed by Y.	4.0%	16.0%	36.0%	44.0%	24.0%	68.0%	8.0%
2. The program you switch to the most is X.	11.5%	30.8%	34.6%	23.1%	23.1%	65.4%	11.5%
3. You spend X% of the time on your computer in program X, Y, and Z.	0.0%	32.0%	32.0%	36.0%	28.0%	60.0%	12.0%
4. X is the program you focus on the longest.	17.4%	21.7%	26.1%	34.8%	17.4%	73.9%	8.7%
5. You feel [more/less] productive when you are focused less. *	23.5%	29.4%	17.6%	29.4%	52.9%	23.5%	23.5%
6. When you feel productive, you spend more time in program X than in Y.	15.0%	20.0%	10.0%	55.0%	30.0%	45.0%	25.0%
7. When you feel unproductive, you spend more time in program X than in Y. *	27.8%	22.2%	22.2%	27.8%	38.9%	38.9%	22.2%
8. You spend more time in Outlook in the [morning/afternoon] than [afternoon/morning].	4.8%	28.6%	33.3%	33.3%	23.8%	66.7%	9.5%
9. You usually work more focused in the [morning/afternoon]. *	26.1%	30.4%	34.8%	8.7%	52.2%	43.5%	4.3%
10. On average, you spend X hours on your computer per work day.	31.8%	18.2%	22.7%	27.3%	45.5%	40.9%	13.6%
11. You feel more productive on days you spend [more/less] time on your computer. *	23.5%	35.3%	11.8%	29.4%	35.3%	64.7%	0.0%
12. You feel [more/less] productive when you send more emails.	14.3%	14.3%	42.9%	28.6%	35.7%	57.1%	7.1%
13. You feel [more/less] productive when you have more meetings.	10.0%	20.0%	50.0%	20.0%	40.0%	50.0%	10.0%
14. You usually feel more productive in the [morning/afternoon].	8.7%	34.8%	39.1%	13.0%	39.1%	47.8%	13.0%
15. You usually take X long breaks (15+ minutes) and Y short breaks (2-15 minutes) from your computer per day. *	21.7%	52.2%	17.4%	8.7%	43.5%	47.8%	8.7%

Design Recommendation C.1: Present multi-faceted correlations using natural language, as users often miss them from reflecting with visualizations.

6.6.6 Potential Impact on Behavior at Work

When we explicitly asked participants if they think they actually changed their behavior during the field study based on the insights they received from using the application, 40.7% reported that they have changed some of their habits based on what they learnt from reflecting about their work. Participants mentioned to be trying to better plan their work (6%), *e.g.*, by taking advantage of their more productive afternoons, trying to optimize how they spend their time with emails (13%), or trying to focus better and avoid distractions (19%).

40.7% of the participants self-reported that they did not change their behavior, either because they did not want to change something (6%) or they were not sure yet what to change (13%). The latter ones mentioned that they needed more time to self-monitor their current behavior and learn more about their habits, and that *PersonalAnalytics* does not offer much help yet in incentivizing or motivating them to change their behavior. In particular, participants stated that the visualizations and correlations were not concrete and actionable enough for knowing what or how to change:

> "*While having a retrospection on my time is a great first step, I gained [...] interesting insights and realized some bad assumptions. But ultimately, my behavior didn't change much. Neither of them have much in way of a carrot or a stick.*" - F42

> "*It would be nice if the tool could provide productivity tips - ideally tailored to my specific habits and based on insights about when I'm not productive.*" - F15

Several participants went on to make specific recommendations for more concrete and actionable support to motivate behavior change. These recommendations ranged from pop-ups to encourage more focused work, to recommend a break from work, all the way to intervening and blocking certain applications or web sites for a certain time:

> "*If [the tool] thinks I am having a productive day, it should just leave me alone and not ask any questions. If I am having an unproductive day and [it] can help me overcome it (*e.g.*, go home and get some sleep) the tool should suggest that.*" - F10

> "*Warnings if time on unproductive websites exceeds some amount, and perhaps provide a way for the user to block those sites (though not forced).*" - F29

When we explicitly asked participants to rate whether or not the 15 personalized insights make them think about or plan their work differently, results indicated that most of the 15 personalized insights are again not actionable enough to foster a behavior change (see results on the right sight of Table 6.3). The five insights with the highest potential (between 40% and 52.9% of participants agreed) are mostly related to work fragmentation and focus on work.

Design Recommendation C.2: Self-monitoring insights often need to be very concrete and actionable to foster behavior change at work.

6.7 Discussion

This section discusses implications that emerged from our study with respect to long-term user engagement, awareness about team-work and collaborations and, ultimately, behavior change.

6.7.1 Design for Personalization

One of our goals was to find out whether the expectations of software developers for a self-monitoring approach are similar or if they are diverging. While existing commercial self-monitoring tools to quantify our lives, such as the **?**, offer only few options for personalization and are still successful at motivating users to live a healthier life [Fritz et al., 2014; Mamykina et al., 2008], our results on self-monitoring at work suggest that personalization is crucial.

In the pilot studies and the field study, participants uniformly expected different measurements to be visualized at different levels of granularity, similar to findings in other areas [Koldijk et al., 2011; McDuff et al., 2012]. These individual expectations might be explained by the very different types of tasks and work that software developers, even with very similar job profiles, have to accomplish [Meyer et al., 2017a]. The ability to customize the measurements that are being captured and how they are visualized is one way to support the personalization. This customizability could not only foster interest in long-term usage, as data relevant to the user is available, but could also reduce privacy concerns that software developers might have.

While many participants were initially skeptical about self-monitoring their work, we received no privacy complaints and most participants (33 of 43) even shared their data with us for the analysis. Almost all participants even went one step further: after a few days of using *PersonalAnalytics* and becoming certain that their data is treated confidentially, they started to comment about possible extensions and additional measures for their self-monitoring at work. This includes more insights about their collaborative activities with other people, as discussed in more detail later in this section, but also adding even more measurements specific to their job as software developers, such as the commits

they make to the version control tool or insights into their patterns of testing and debugging code.

While it might seem surprising that developers requested many development-unrelated measures for self-monitoring their work, this can be explained by the amount of time they spend with development related activities, on average between 9% and 21%, versus other activities, such as collaborating (45%) or browsing the web (17%) [Gonçalves et al., 2011; Meyer et al., 2017a]. As most study participants (84.6%) were interested to continue using *PersonalAnalytics* after the study had ended, we concluded that the initially identified design elements to support various individual needs, actively engage the user, and enable more multi-faceted insights are valuable for self-monitoring at work.

6.7.2 Increased Engagement through Experience Sampling

As noted in previous research, many self-monitoring approaches suffer from an extremely low user engagement with the data [Collins et al., 2014; Huang et al., 2016; Kim et al., 2016]. For example, RescueTime, which visualizes the captured data on a dashboard in the browser, was found to be used only a few seconds per day (mean=4.68 ±12.03) [Collins et al., 2014]. Similar to the reports in our field study, participants' reasons for this low engagement might be that users forget about the availability of the data visualizations. A simple and periodic reminder, *e.g.*, to let users know that there is a new summary on the work week, might increase the engagement with these visualizations and dashboards. Recently, researchers have explored how adding an ambient widget and presenting a summary of the captured data always visible on the user's screen can increase the engagement with the data (*e.g.*, [Collins et al., 2014; Kim et al., 2016; Whittaker et al., 2016]). For example, the ambient widget by Kim et al. [2016] increased the use of RescueTime to about a minute a day.

In this paper, we assessed another approach, namely a periodic pop-up to self-report productivity. Our findings show that the self-report helped users to quickly reflect on how efficiently they spent their time, which then also resulted in an increased engagement. Our results show that using experience sampling is a feasible method to manually collect data that is difficult to capture automatically

and is (mostly) appreciated as long as users have a benefit from self-reporting, *e.g.*, by getting additional or more fine-grained insights. It is up to future work to determine how long the positive effects of self-reporting or ambient widgets lasts, whether users might at some point loose interest after having learnt 'enough' about their work, and whether it might be beneficial to only include these features in certain time periods. More research is required to understand how this can be generalized to other domains.

6.7.3 Actionability for Behavior Change

Most health and sports tracking systems have been shown to foster positive behavior changes due to increased self-awareness. In our study, 40.7% of the participants explicitly stated that the increased self-awareness motivated them to adapt their behavior. While motivating changes in behavior was not a primary goal, the study gave valuable insights into where and how self-monitoring tools at work could support developers in the process. The very diverse set of insights in *PersonalAnalytics* that participants wished for, made it more difficult to observe a specific problem behavior and define a concrete, actionable goal for a behavior change, which is a basic requirement for starting a change according to the theory of behavior change process TTM [Prochaska and Velicer, 1997]. Rather than just enabling an increased self-awareness, it might also be important to provide users with concrete recommendations for active interventions and alerts when certain thresholds are reached. Participants suggested to block distracting websites after the user spent a certain amount of time on them, or to suggest a break after a long time without one, similar to what was recently suggested [Agapie et al., 2016; Epstein et al., 2016a]. At the same time, not all insights are actionable as developers sometimes have little power to act on an insight, similar to what Mathur et al. [2015] found from visualizing noise and air quality at the workplace. As an example, most developers can likely not just stop reading and responding to emails. Another extension to possibly make insights more actionable is to let users formulate concrete behavior change goals based on the insights they make from using the retrospection and experience sampling component. For example, a user could set a goal to regularly take a

break to relax or to have an empty email inbox at the end of the day. This goal setting component could leverage experience sampling further and learn when and how users are interested and open to receive recommendations of how to better reach their goal.

Approaches aiming to foster long-term behavior changes need to offer means to actively monitor and maintain a behavior change [Prochaska and Velicer, 1997] and help avoiding lapses, a frequent reason for abandoning behavior change goals [Agapie et al., 2016]. In the future we plan to experiment with and evaluate these different forms of how insights could be improved to make them more actionable, and then evaluate the longer-term impact of *PersonalAnalytics* on software developers' actual behavior at work.

6.7.4 Benchmarking

A re-occurring feedback by participants was the wish for a way to benchmark their work behavior and achievements with their team or other developers with similar job profiles and to improve their work habits based on the comparisons with others, similar to what was previously described by Wood [1989]. Given the privacy concerns at work, adding such a component to the self-monitoring for work could, however, severely increase pressure and stress for users who are performing below average. Also, given our participants' interest in a high variety and large set of work related measures indicates that even within one domain—software developers in our case—users might work on fairly different tasks and that it might be impossible to find a 'fair' set of measures for comparing and benchmarking individuals. More research is needed to examine how and in which contexts such a social feature might be beneficial as well as which aggregated measures might be used for some sort of comparison without privacy concerns. For example, one could anonymously collect the data related to developers' work habits, such as fragmentation, time spent on activities, and achievements, combine them with job profile details and then present personalized insights and comparisons to other developers with a similar job profile. One such insight could be to let the developer know that others spend more time reading development blogs to educate themselves or that they usually have less meetings in a work

week. Besides having anonymous comparisons between developers, it could further be beneficial to let users compare their work habits with their previous self, *e.g.*, from one month ago, and enable them to reflect on how their behaviors change over time. Although research has shown that benchmarking features in physical activity trackers foster competition with peers to be more active [Fritz et al., 2014; Rooksby et al., 2014], additional research is needed to determine whether they also lead to a positive behavior change at the workplace.

6.7.5 Team-Awareness

Even though most insights available within *PersonalAnalytics* appear to be about each user's own work habits, some insights also reveal details about the individuals' collaboration and communication patterns with their team and other stakeholders. These are, for example, insights about their meeting, email, instant messaging, social networking, and code review behavior. Nonetheless, participants were interested in even more measures, especially with respect to revealing (hidden) collaboration and communication patterns within their teams. Having detailed insights into how the team coordinates and communicates at work could help developers make more balanced adjustments with respect to the impact their behavior change might have on their team. For example, being aware of co-workers' most and least productive times could help to schedule meetings at more optimal times, similar to what Begole et al. [2002] suggested for teams distributed across time zones. Related to an approach suggested by Anvik et al. [2006] where work items and bug reports were automatically assigned to developers based on previously assigned and resolved work items, it could be beneficial for improving the coordination and planning of task assignments by also taking into account each developer's current capacity and workload. Being more aware of the tasks each member of the team is currently working on and how much progress they are making could also be useful for managers or team leads to identify problems early, *e.g.*, a developer who is blocked on a task [Jakobsen et al., 2009] or uses communication tools inefficiently [Storey et al., 2017], and take appropriate action. A similar approach, WIPDash, has been shown to improve daily stand-up meetings by providing teams with shared dashboard

summaries of work items each developer was assigned to and has completed, as these dashboards increase the awareness about each developer's progress on tasks [Jakobsen et al., 2009]. Visualizing the current productivity and focus to co-workers could prevent interruptions at inopportune moments, where resuming the interrupted task might be more costly than at a moment of low focus. To streamline inopportune interruptions at work, Züger et al. [2017] suggested to visualize the current focus to the team by using a "traffic light like lamp".

As the envisioned additions and extensions to *PersonalAnalytics* might increase an individual's productivity, they might negatively affect the overall team productivity or the collaboration within teams. For example, a developer who is stuck on a task cannot ask a co-worker for help that blocks out interruptions. This is why self-monitoring tools for teams at work could not only motivate a collective improvement of the team-productivity, but also help to monitor the success and impact of these changes on other stakeholders. Future work could explore how self-monitoring at work supports team collaboration, by analyzing collaboration practices within development teams and comparing them to other teams. This work could be based on the Model of Regulation, recently introduced by Arciniegas-Mendez et al. [2017], as it helps to systematically evaluate and understand how teams self-regulate their own tasks and activities, other team-members, and how they create a shared understanding of their project goals.

6.8 Generalizability and Limitations

We focused our work on one type of knowledge workers, software developers, to gather insights into one work domain before generalizing to a broader range of knowledge workers in the future. Software developers have been referred to as the *knowledge worker prototype* as they are often not only the first ones to use and tweak tools, but also have lower barriers for building and improving tools themselves [Kelly, 2008]. While software developers experience extensive interaction and collaboration with co-workers through their computer use, we believe that many of the observations made from building and evaluating *Per-*

sonalAnalytics with developers are also helpful for self-monitoring tools in other work domains, especially since the studied features and most tracked measures can be re-used in or easily ported to other domains.

The main threat to the validity and generalizability of our results is the external validity, due to the selection of field study participants that were all from the same company and had limited gender diversity. We tried to mitigate these threats by advertising the study and selecting participants from different teams in the company, at different stages of their project, and with varying amounts of experience. Participants tested *PersonalAnalytics* over a duration of several weeks and were studied in their everyday, real-world work environment and not in an experimental exercise. Moreover, the development of the application was designed together with participants from three other companies of varying size, reducing the chance that we built an application that is just useful for software developers at one company. Although our findings shed light on how awareness and engagement can be increased, it is not clear how *PersonalAnalytics* affects software developers using it over longer than the three-week period studied. We are aware that there is a certain self-selection bias towards participants who are in general more willing to quantify various aspects of their life, and use the collected data to increase their awareness.

6.9 Conclusion

One way to improve the productivity and well-being of knowledge workers is to increase their self-awareness about productivity at work through self-monitoring. Yet, little is known about the expectations of and experience with self-monitoring at the workplace and how it impacts software developers, one community of knowledge workers on which we focused. Based on previous work, an iterative development process with 5 pilot studies and a survey with 413 developers, we factored out design elements that we implemented and refined with *PersonalAnalytics* as a technology probe for self-monitoring at work. We then evaluated the effect of these design elements on self-awareness of patterns of work and productivity and their potential impact on behavior change with 43

participants in a field study, resulting in design recommendations.

We found that experience sampling, using minimal-intrusive self-reporting, and the retrospective summary of the data enhances the users' engagement and increases their awareness about work and productivity. Participants reported that by using our self-monitoring approach, they have made detailed observations into how they spend their time at work collaborating or working on tasks, when they usually feel more or less productive, and sort out misconceptions they had about their activities pursued at work, such as spending a surprisingly high amount of time collaborating with others via email. Our work provides a set of design recommendations for building self-monitoring tools for developers' work and possibly other types of knowledge workers. We discuss potential future work to further increase engagement with the data and to enhance the insights' actionability by providing users with recommendations to improve their work, by adding social features to motivate users to compete with their peers, and by increasing the team awareness to help teams reduce interruptions, improve the scheduling of meetings, and the coordination of task assignments.

6.10 Acknowledgements

The authors would like to thank the study participants and the anonymous reviewers for their valuable feedback. This work was funded in part by Microsoft, NSERC and SNF.

7

Enabling Good Work Habits in Software Developers through Reflective Goal-Setting

André N. Meyer, Gail C. Murphy, Thomas Zimmermann, Thomas Fritz
Accepted in the 2019 IEEE Transactions on Software Engineering Journal, Contribution: Study design and execution, participant recruitment, data collection, data analysis, and paper writing

Abstract

Software developers are generally interested in developing better habits to increase their workplace productivity and well-being, but have difficulties identifying concrete goals and actionable strategies to do so. In several areas of life, such as

the physical activity and health domain, self-reflection has been shown to be successful at increasing people's awareness about a problematic behavior, motivating them to define a self-improvement goal, and fostering goal-achievement. We therefore designed a reflective goal-setting study to learn more about developers' goals and strategies to improve or maintain good habits at work. In our study, 52 professional software developers self-reflected about their work on a daily basis during two to three weeks, which resulted in a rich set of work habit goals and actionable strategies that developers pursue at work. We also found that purposeful, *continuous* self-reflection not only increases developers' awareness about productive and unproductive work habits (84.5%), but also leads to positive self-improvements that increase developer productivity and well-being (79.6%). We discuss how tools could support developers with a better trade-off between the cost and value of workplace self-reflection and increase long-term engagement.

7.1 Introduction

Software developers are motivated to develop better habits to improve their productivity and well-being at work [Li et al., 2015; Meyer et al., 2017b; Sharp et al., 2009]. It is therefore desirable to gain a better understanding of what *good* work habits and behaviors are, and how we can support developers with the identification of self-improvement opportunities to build better and maintain good habits at work. Prior research has examined developers' existing work habits, specifically the time they spend on various activities at work (*e.g.*, [Astromskis et al., 2017; Gonçalves et al., 2011; Meyer et al., 2017a; Singer et al., 2010; Xia et al., 2017]), their organization of work into tasks (*e.g.*, [González and Mark, 2004]), and causes of fragmented work (*e.g.*, [Chong and Siino, 2006; Czerwinski et al., 2004; Mark et al., 2005; Meyer et al., 2017a; Parnin and Rugaber, 2011]). Recently, researchers have also looked into the attributes and habits of *great* software developers [Baltes and Diehl, 2018; Li et al., 2015; Sharp et al., 2009]. They found that one key trait of successful developers is growth orientation, which means that they are constantly learning and striving to change their behavior to increase efficiency at work.

Goal-setting is one way to foster behavior change, since it allows individuals to define a target or outcome, and make progress towards their goal [Locke and Latham, 1990, 2002]. In the context of this work, *goals* refer to desired target or outcome habits that developers set for themselves, to improve productivity and well-being at the workplace. *Strategies* refer to the system they employ to make progress towards and eventually reach their goals. However, identifying concrete and relevant goals can be challenging, which is why an active area of research is investigating how self-reflection can help individuals to get insights into positive and negative habits, and support them with the identification of meaningful goals that motivate positive behavior changes [Brockbank and McGill, 2007; Kersten-van Dijk et al., 2017,?; Sharot et al., 2011]. This includes mostly personal areas of life, such as health [Gasser et al., 2006; Monkaresi et al., 2013], sleep quality [Daskalova et al., 2016; Lee et al., 2017], students' learning behavior [Johnson and White, 1971; Morisano et al., 2010; Travers et al., 2015], and physical activity [Herrmanny et al., 2016; Klasnja et al., 2009; Munson and Consolvo, 2012]. Research more specific to knowledge workers' work habits investigated the effects of self-reflection on task completion [Amabile and Kramer, 2011; Claessens et al., 2010], time management [Pammer et al., 2015], and detachment from work [Kocielnik et al., 2018; Williams et al., 2018].

While developers generally want to play an active role in setting their own goals for work [Couger et al., 1990; Enns et al., 2006; Perry et al., 1994b], we have not been able to find prior work that investigated goals developers set to improve work habits and productivity. This is why we wanted to study the goals that developers set for themselves to improve and maintain good habits at work, the strategies they pursue to achieve those goals, and the impact their goal-setting has on productivity and well-being. Even though self-reflection has previously been shown to have great potential to foster goal-identification, developers rarely reflect on or review their work in practice [Baltes and Diehl, 2018]. Hence, we further aimed to examine whether encouraging developers to self-reflect continuously on work, results in meaningful insights about work and leads to any work habit goals and -improvements. In particular, our work seeks to answer the following research questions:

RQ1: Which types of goals do developers set for themselves to improve and maintain good work habits?

RQ2: What are strategies that help developers make progress towards, and achieve their goals?

RQ3: What is the potential impact of reflective goal-setting on developers' goal-identification, goal-achievement monitoring, and work habits?

To investigate these research questions, we combined self-reflection and goal-setting to design a reflective goal-setting study, inspired by the Personal Software Process (PSP) by Humphrey [1996] and diary studies in other areas of research. Our study prompts participants on a daily basis to reflect on their work, and asks them to set concrete goals and actionable strategies for improving their work habits. 52 professional software developers completed our study and reflected for two to three work weeks.

Our reflective goal-setting study resulted in a rich set of work habit goals and strategies that we analyzed. They can be broadly categorized into improving time management, avoiding deviations from planned work, improving the impact on the team, maintaining work-life balance, and continuous learning. We found that *continuous* self-reflection can be an important step towards productive self-improvements in the workplace, since participants stated that it supports the identification of goals (80.8%) and actionable strategies (83.3%). The daily self-reflections not only increased developers' awareness about work habits, progress and achievements (84.5%), but also led to productive (short-term) behavior changes (79.6%). As a result, while initially being skeptical towards "journaling" their work, most participants (96.1%) stated afterwards that they could imagine to continue self-reflecting on a regular basis. Few participants, however, mentioned that constantly self-reflecting may increase pressure to always perform well and thus, could turn into a burden without tool support that would make self-reporting more convenient. Overall, we conclude that *continuous* reflective goal-setting can enable developers to improve and maintain good work habits. We discuss these results with regards to prior work on self-reflection with other types of knowledge workers, and how tools could support developers with their

reflective goal-setting and how they might foster long-term self-reflection.

Our contributions are (1) a set of developers' good work habit goals and strategies to improve productivity based on a field-study, and (2) insights into the use and value of continuous reflective goal-setting, and its ability to support developers with the identification, monitoring and maintenance of good work habits that improve productivity and well-being at work.

7.2 Related Work and Background

Work related to our research can be broadly categorized into research that examined developers' work and productivity, what productive work habits are, and how to foster these with goal-setting and self-reflection. To avoid repetitions in this thesis, we present and summarize the related work in Section 1.6 of the synopsis.

7.3 Study Design

To answer our research questions, we conducted an *in situ* study at 10 software development companies of varying size. We collected data from 52 professional software developers using a structured reflective goal-setting process, a self-experimentation framework that we developed based on previous work in other research areas.

Reflective Goal-Setting Our reflective goal-setting study is based on successful self-reflection and goal-setting approaches from other areas of life, in particular the health domain where self-experimentation is researched much more extensively. The core of our study are the daily self-reflection questionnaires (Table 7.1), a morning questionnaire participants were asked to answer before they start their work, and an afternoon questionnaire they answer at the end of their workday. The self-reflection questionnaires are based on the same stages of reflective goal-setting as identified by Travers et al. [2015] and summarized in Table 7.2.

Table 7.1: Daily Self-Reflection Questions.

MORNING QUESTIONNAIRE
Please answer the following question after planning your workday and before starting with your work
Q1: What do you plan to achieve today? This could include tasks and other things you want to make progress on at work. *[5 empty textboxes]*
Previously, you've described the following goal to improve/maintain good habits at work: *[copy previous goal here]*
AFTERNOON QUESTIONNAIRE
Q1: Please rate whether you achieved what you set out to do this morning for your following items *[show planned items from morning questionnaire, options: didn't work on it, made progress on it, completed it]*
Q2: What was the best work-related thing that happened to you today at work?
Q3: Was there anything else that you achieved but didn't plan for in the morning?
Q4: Was there something that made it difficult to achieve what you planned to do?
Previously, you've described the following goal to improve or maintain good work habits: *[copy previous goal here]*
Q5: Did you actively try to achieve the goal? *[options: yes, no]*
Q6: *[show if 'yes' in Q5]* How did you carry out your strategy? Did it positively influence your workday?; *[show if 'no' in Q5]* Why not?
Q7: Is there a goal that you would like to set for yourself that could help you to improve or maintain good work habits? You can revise your existing goal, set a new goal, or keep the same goal (copied here for convenience). Try to follow the SMART goals principle, i.e. think about a goal that is specific, realistic to achieve and matters to you. It might be helpful if you also think about when you want to achieve that goal (time-bound) and how you could measure your progress towards it. *[copy previous goal into textbox if available]*
Q8: What's the first step you will take towards reaching this goal?
Q9: Did any other goals come to mind today that you would like to mention?
Q10: How did you approach assessing your achievements and the progress you've made today?

Instructions are in brackets. Where not explicitly stated, participants' response was collected with a textbox.

Table 7.2: Stages of Reflective Goal-Setting [Travers et al., 2015].

Stage 1	increase self-awareness
Stage 2	selecting suitable growth goals
Stage 3	visualizing future growth goal behavior and techniques
Stage 4	identifying tools and techniques to apply goal
Stage 5	putting growth goals into practice with ongoing reflection

The **morning questionnaire** asked participants to state the five most important things they want to achieve that day (stages 1 and 5 in Table 7.2). Since work by Altmann and Trafton [2002] showed that prospective goal-encoding, *i.e.*, the "action of looking ahead mentally to determine how to proceed" is valuable for successfully achieving goals, the morning questionnaire also showed the previously set goal to remind participants to pursue it on the workday ahead.

The **afternoon questionnaire** consisted of three parts: 1) *Where am I?* (stage 1 in Table 7.2), 2) *Where do I want do go?* (stage 2), 3) *How do I get there?* (stages 3 and 4). Previous work substantially influenced how we prompted participants to reflect on work, and define goals and actionable strategies. For example, work by Brockbank and McGill [2007] has shown that reflection provides a valuable feedback mechanism that supports people to monitor and measure their progress towards a goal. Hence, the *first* part included a reflection step where participants rated their progress on the five items they stated they want to achieve in the morning questionnaire. In addition, participants reflected on aspects that were positive or made it difficult to make progress, and any other unplanned items they made progress on. In case they had defined a goal and strategy on the previous workday, the questionnaire prompted participants to reflect on their achievement of the goal and whether the strategy worked. As previous work has shown that writing down goals and committing on a single goal only enhances goal achievement [Matthews, 2007; Wiseman, 2007], the *second* part of the questionnaire prompted participants to commit to and write down a goal for their next workday, using the SMART goals approach [Doran, 1981] that was shown to result in more specific and more concrete goals [Lee et al., 2014, 2015]. Since goals are rarely perfect from the beginning and they can change over time, participants could alternatively also revise an existing goal. [Locke and Latham, 1990]. In the *third* part, we asked participants to describe a concrete first step they plan to take towards reaching the goal, as this was previously shown to increase the likelihood of actually changing one's behavior towards a goal [Allen, 2015; Clear, 2018; Prochaska and Velicer, 1997]. Note that we did *not* provide participants with any examples of goals or strategies, to avoid biasing them. Every third day, participants were asked how they assessed the

progress they made towards their achievements.

Study Setup Before the study, we emailed participants a document explaining the study objectives and procedure, explained the SMART goals approach, asked them to sign a consent form as well as to answer a pre-study survey with questions on demographics, their existing goal-setting and -measuring practice and how they plan their work. Participants were then asked to pursue their work as usual for the duration of the study, while answering the morning and afternoon questionnaires timely. We sent two email reminders per workday, one at 8am for the morning questionnaire, and the other one at 4pm for the afternoon questionnaire. We asked participants to answer the daily questionnaires for two to three weeks, to avoid boredom or fatigue and ensure high quality responses. We also logged the time it took participants to answer each daily questionnaire. At the end of the study, participants were asked to answer the final survey, in which we asked participants to rate their agreement with statements, or open questions on their awareness, goal-setting, goal-measuring, and behavior change before and during the study. The final survey also asked participants about the value and impact of reflective goal-setting, if they could imagine integrating it into their work routine, and how it could be best assisted with tool support. After completing the final survey, participants received a 50 USD Amazon gift certificate as a compensation for their efforts.

Study Test-Run To test-run and gather qualitative feedback in advance, we ran a pilot study with three computer science graduate students. The feedback helped us to fine-tune the survey questions and formulations. For example, participants in the pilot study suggested to show the previous workday's goal again in the morning questionnaire, and to introduce the SMART goals approach prior to the daily questionnaires.

Participants We recruited 59 participants through professional and personal contacts from 10 companies, ranging from startups to large multi-national corporations in the software industry. Developers were invited to join the study either

after giving research talks at the respective companies or by spreading the study description via our network on social media. Participants could freely decide to participate and were not enforced by their employers. We discarded data from 7 participants that answered the daily self-reflection surveys for less than one week, or if the time spent on the daily questionnaires was regularly shorter than one minute each–too short for being a meaningful reflection. Of the remaining 52 participants, 7 were female and 45 were male. Our participants had an average of 8.2 (±6.7, ranging from 1 to 24) years of professional development experience, and were working in different roles: 45 identified themselves as individual contributors and 7 as developers in a leading position (*i.e.*, development lead or manager). Participants resided in the US, Canada, Brazil or Switzerland.

Data Analysis In total, we collected 605 self-reflections (*i.e.*, afternoon questionnaires) from 52 participants. On average, they completed 11.6 (± 2.7) self-reflections each, which equals 11.6 days of self-reflection. We qualitatively analyzed the collected data, and identified themes by finding commonalities and key concepts from performing a Thematic Analysis [Braun and Clarke, 2006]. To analyze the responses, we first open coded participants' self-reported goals, using a quote-by-quote strategy where multiple codes could be assigned to each quote. Responses that could not distinctively be mapped to a code where discussed with the other authors. To identify high-level goal categories, we discussed the resulting codes and their relationships in multiple team discussions as well as one card sort session. During the whole process, we heavily relied on the quotes and consulted them regularly for additional context and details about the identified relationships. The whole process was iterative, meaning that whenever the discussions resulted in updates to the open coding categories, we did another iteration.

7.4 Developers' Work Habit Goals and Strategies (RQ1, RQ2)

An overview of the work habit goals and strategies that developers set for themselves is presented in Table 7.3. Overall, the identified goals can broadly be grouped into five categories: (G1) improve time management, (G2) avoid (self-induced/external) deviation from planned work, (G3) improve impact on the team, (G4) maintain work-life balance, and (G5) learn continuously. The majority of developers' goals describe **continuous behaviors and desirable habits** they want to develop, rather than momentary goals that have a defined outcome or result. While many of these goals could apply to any knowledge worker, several goals are very **specific to software development** (marked with [SE] in the table). The goal categories are not necessarily disjunct categories: For example, improving time management (G1) can impact what developers consider to be a deviation from planned work (G2). The reflective goal-setting study allowed participants to not only identify work habit goals, but also experiment with different strategies to reach them. In the evening questionnaires, participants also reflected on whether the strategy that they set on their previous workday worked. After coding developers' goals, we analyzed developers strategies and summarized the successful ones in Table 7.3. Most strategies describe *concrete and actionable* habits and routines that participants usually wanted to perform on a daily basis or on multiple occasions each day.

In the following, we discuss each of the five goal categories and participants' strategies to reach them, using participants' examples and quotes. The numbers in parentheses are counts for the number of developers that set one or multiple goals of each category. Since the identified goals are based on what participants identified themselves, the reported numbers should only serve to give a feeling about how prevalent each goal is, as actual numbers might be higher. Goal categories are enumerated with G, strategies with S, and participants with P.

7.4.1 Improve time management (G1)

Plan workdays in advance (36) A goal that the majority of the participants set is to improve their time management by planning their workday in advance, either on a daily or weekly basis:

> "*I should make a TODO list for today, not a general TODO list for 'at some point'.*" - P33

Most strategies to reach this goal include common planning methodologies, such as *maintaining a personal task list (S1)*, *reserving time to work on important tasks in the calendar (S2)*, and *planning buffers (S3)* for unplanned tasks or issues. To actually apply these strategies, and not forget performing them after just a few days, participants tried to *develop a daily routine (S4)* of planning their next workday the evening or morning before work, or by setting an *alarm or reminder (S5)*:

> "*I scheduled an event in my calendar to remind myself to complete a code review. I prioritized this over other work. It was also easier to start on this since it is schedule right after my team's daily stand up meeting which means I am not in the middle of a task.*" - P4

> "*When I think a task takes 2 days to finish, add another day for unforeseen issues unless I am certain.*" - P36

Make progress on most important tasks first (33) Developers also want to continuously make progress on their most important tasks. To that purpose, they applied strategies such as constantly *reviewing their priorities (S7)*, and regularly *reflecting on the progress (S8)* made on their tasks:

> "*It occurs to me that I need to quick-scan my inbox, and look at my calendar, before planning the day. It will be important that I _scan_ the inbox, and not respond to anything. If I do that, I'll get stuck into answering mails, and not plan.*" - P33

> "*At the end of the day or beginning of [the] next day, [I want to] reflect why not all goals could be fully completed, and if so, why and what could be improved.*" - P12

These two strategies helped participants to stay aware of their most urgent and most important tasks, which allowed them to plan for an efficient workday and work more systematically.

Make better use of work item tracking tools (6) For their development work, participants further mentioned they want to better and more actively use their work item (WI) tracking tools, by *updating their progress (S8)*, *documenting new findings from investigations in the code (S10)*, and by *defining subtasks for each work item (S6)*:

> "*Every morning, breakdown user stories into at least 2 sub-tasks before starting or continuing work on it. If it is an existing story, update the tasks based on any new findings/discussions.*" - P4

7.4.2 Avoid (self-induced/external) deviation from planned work (G2)

Developers are also interested in forming habits that make it easier to follow through with their planned workday (G1) and react better in case things deviate from the plan.

Better handle urgent/unplanned issues/bugs (29) Developers want to improve how they handle unplanned, urgent issues and bugs, such as a production issue or a build break, which can occur frequently in software development. Rather than always starting to work on the unplanned issue immediately, they first want to *review their priorities (S7)* before consciously deciding how to act:

> "*Before doing anything, I ask myself 'How does this immediately contribute to what I need to deliver this month?' and 'Can this be delayed or even avoided?'.*" - P19

Deferring unplanned tasks or bugs can effectively help developers to better balance and make progress on their planned development tasks, instead of being constantly sidetracked. Other strategies are to *plan buffers (S3)* and to *set concrete upper limits (S11)* for working on unplanned issues:

> "*Spend less than 1 hour a day in activities that are not part of the day's [plan]*" - P25

In case participants decided to stop working on the current task and start working on the unplanned issue, many wanted to update their *workday plan (S9)* to reschedule their planned tasks.

Improve focus: reduce distractions and interruptions (33) Similar to other knowledge work domains, developers aim to limit their exposure to aspects that reduce their focus, to be able to concentrate for longer chunks of time on planned work. Specifically, developers' goals primarily aim to reduce the amount of external interruptions from co-workers and self-distractions:

> "*Reduce the time spent on checking mails and [IM] channels and restrict it to specific hours like once in morning and at end of the day.*" - P41

Strategies to reduce external interruptions vary, ranging from changing *how* developers organize their work (*e.g.*, *timeboxing work (S12)*) to *where* they work (*changing the location (S16)*):

> "*Scheduled daily time block from 8AM to 10AM in my calendar: Stayed at home. Stayed offline until 10AM (not completely offline, but never checked E-Mail or Slack etc. before my time block was over.)*" - P8

> "*Reduce meeting time or club meetings back to back to have bigger focus time blocks on my calendar.*" - P41

Developers further customized their communication tools, either by *disabling notifications (S13)* at certain times, or by *reducing the number of communication channels (S14)*).

> "*I have [...] notifications turned off completely in Outlook and Slack. I don't run anything such as Facebook or YouTube in the background or on a second screen.*" - P6

Reducing self-distractions requires a lot of self-control. To reduce the temptation to regularly check social media or news websites, few participants installed browser extensions to *block work unrelated websites (S17). Preparing their environment for focused work (S18)*, such as hiding the smartphone or filling the water bottle, further allowed participants to work focused for longer blocks, but also reduce multi-tasking, which was previously shown as a source of stress and quality issues [González and Mark, 2004; Mark et al., 2008b].

Balance coding time (10) Another reason why developers want to avoid deviations form work is to ensure they can spend enough time with coding, by balancing activities and *setting limits (S11)* or by *blocking a specific amount of time for coding in the calendar (S2).* For example, some aim to spend a minimum amount of time on coding each day, want to reduce time spent with bug triaging, or want to better balance the time spent working on new features versus fixing bugs:

> "*Allocate a reasonable amount of the day for bug triaging of test runs (∼2 hours max).*" - P14

The importance of finding a good balance between coding and other work was recently shown to increase developers' job satisfaction [Meyer et al., 2019].

7.4.3 Improve impact on the team (G3)

Besides making progress on their own tasks, developers are also interested in improving their team impact and collaboration.

Be a better colleague (12) Participants also wanted to become a better colleague. However, both, the goals and strategies were often not concrete, and participants had troubles to reach them. For example, participants mentioned that they wanted to become a better pair programmer, or better understand their influence within the team, without being able to clarify what this goal meant or how they could reach it. Two strategies that developers successfully applied to become a better colleague were to *delegate important tasks to their co-workers (S21)* instead of always keeping them for themselves, and to actively *share their knowledge and learnings (S22)* with co-workers:

> "*Stop writing code trying to prove to the others that I can do complex stuff.*" - P37

Help co-workers at specific times (5) Even though developers want to reduce external interruptions to have longer times of focus, they also want to support and help their co-workers with questions or in case they are blocked on a task. To reduce the exposure they have to external interruptions, they ask *co-workers to*

adapt (S15), *e.g.*, by asking them to schedule time for questions in the calendar in advance, or by *changing their location (S16)*, *e.g.*, by moving to a quiet place in the office or working from home:

> "*Get people to book time with me when they need my help instead of helping them immediately.*" - P1

Do more code reviews (9) More specific to software development, developers aim to perform code reviews more frequently, to provide continuous feedback to their team and to help increasing the software quality. To achieve these goals, they want to develop daily or weekly *routines (S4)* or *set reminders (S5)*:

> "*Complete one code review every day before lunch.*" - P4

Keep documentation updated (8) Developers also set goals to become more diligent at writing documentation and keeping it updated. With this goal, they not only aim to make their own work easier and more efficient, by remembering previous changes and decisions, but also to better support their co-workers. To achieve their goal, participants either wanted to *document consistently (S10)*, *e.g.*, immediately after an important change to the code base or infrastructure, or to *develop a routine (S4)* of documenting when they had time to spare. This goal is not specific to development documentation only, but also includes logging progress on tasks, reporting bugs they identify in the coding process, or improvements they make to the development process (*e.g.*, release process).

Work more independently (7) Developers also want to work more independently, to not have to interrupt their co-workers all the time to ask for guidance or help. Participants who set this goal were often junior developers who tried to approach problems more systematically to reach their goal. However, when they reflected about whether the strategy helped, most reported that it was not actionable enough. Few participants reported that they then seeked out to ask co-workers for mentorship on how they could become more independent, who suggested to *document the thought process (S10)* during the investigation of the problem, which participants tried and reported that it helped them.

7.4.4 Maintain work-life balance (G4)

Developers further set goals to maintain a good work-life balance.

At work: stay motivated and fresh (11) At work, developers want to have more regular and productive work hours to stay motivated and fresh. To achieve their goal, they regularly take *short social breaks (S23)* and use these breaks to have meaningful conversations with co-workers or learn from them. Participants emphasized these breaks must be short, since they otherwise eat into their work or spare time. Participants further optimized their work by *reducing their work hours (S24)*, *e.g.*, leaving earlier and accepting there will always be work left to do, and by *working more regular hours (S25)* and avoiding night shifts or weekend work.

Outside of work: better detach from work (7) Outside of work, developers want to be able to enjoy their spare time more and regenerate for work, by better detaching from work and not thinking about it all the time. We summarized their strategies to achieve the goal as *pursuing a more sustainable lifestyle (S26)*. They include exercising more, sleeping longer and better, eating healthier, meditating and journaling:

> "*Don't think about work outside of work. Try to be satisfied with the daily work and come back fully rested.*" - P11

Three participants also wanted to compare the data they collected on their workplace detachment (*e.g.*, sleep or exercise data) with their perceptions of productivity or efficiency to spot patterns and gain ideas for further self-improvements, but they had difficulties to follow-through and, hence, asked for tool support.

7.4.5 Learn (G5)

Learn/improve specific development technology/transferable skill (16) Developers also set learning goals, either for learning the basics of a specific technology, such as a framework or programming language, or of a transferable development skill, such as architectural design, design patterns or how to troubleshoot issues. In contrast to the other goal categories (G1-G4), learning goals do not describe a continuous behavior developers want to develop, but a specific goal to reach within a concrete timeframe, which is to better understand or improve their knowledge on a topic:

> "*I want to know the basics of redux-saga within 7 days.*" - P11

To achieve their goal, participants' strategies included developing a *learning routine (S4)* of always seizing opportunities to learn when they had a bit of time, or *reserving re-occurring time blocks for learning in the calendar (S2)* . Developers also wanted to better understand the product they were part of and its long-term vision, and attempted to reach the goal by asking more senior *co-workers for help (S27)*, and by *asking more specific questions (S28)*.

Improve myself (7) Besides learning more about development topics, developers want to continuously improve themselves, for example by learning to work and communicate more efficiently. To achieve their goal, many participants developed a habit of *regularly reflecting about their goals (S8)*, to consider the progress they made towards them, and then decide to take specific actions for self-improvements:

> "*Go over todo list in the evening commute to check which todos were completed. If not, think about why they are not yet done, write down the reason once a week, go over all the reasons to check for patterns.*" - P12

Table 7.3: Developers' Work Habit Goals and Strategies.

Goal	Strategies	Goal-Achievement Monitoring
Improve time management (G1)		
Plan workdays in advance (36)	Maintain personal task list (S1), block time for tasks in calendar (S2), plan buffers (S3), develop a routine (S4), set reminders (S5), create subtasks (S6)	updates to task list and calendar
Make progress on most important tasks first (33)	Review priorities (S7), reflect on progress towards goals/tasks (S8), update plan (S9)	prioritized task list; monitor task switch behavior; self-reflection
Make better use of work item tracking tools (6) [SE]	Reflect on progress towards goals/tasks (S8), document findings & learnings (S10), create subtasks (S6)	updates to WI tracker
Avoid (self-induced/external) deviation from planned work (G2)		
Better handle urgent/unplanned issues/bugs (29) [SE]	Review priorities (S7), plan buffers (S3), set limits (S11), update plan (S9)	prioritized task list; calendar; monitor task switch behavior
Improve focus: reduce distractions and interruptions (33)	Timebox/group work (S12), disable notifications (S13), reduce communication channels (S14), ask co-workers to adapt (S15), change location (S16), block apps/websites (S17), prepare environment for focus (S18), focus on single task (S19), complete tasks (S20)	monitor activity and task switch behavior, notifications (computer & phone), and location; self-reflection
Balance coding time (10) [SE]	Set limits (S11), block time for tasks in calendar (S2)	monitor coding time
Improve impact on the team (G3)		
Be a better colleague (12)	Delegate important tasks (S21), share knowledge/findings (S22)	monitor tasks assigned to others in WI tracker
Help co-workers at specific times (5)	Ask co-workers to adapt (S15), change location (S16)	reduced task switching
Do more code reviews (9) [SE]	Develop a routine (S4), set reminders (S5)	number of code reviews
Keep documentation updated (8) [SE]	Document findings & learnings (S10), develop a routine (S4)	regular updates to documentation
Work more independently (7)	Document findings & learnings (S10)	
Maintain work-life balance (G4)		
At work: stay motivated and fresh (11)	Take short social breaks (S23), reduce work hours (S24), work more regular hours (S25)	monitor breaks and work hours
Outside of work: better detach from work (7)	Pursue sustainable lifestyle (S26)	self-reflection; monitor activities outside of work
Learn (G5)		
Learn/improve specific development technology/transferable skill (16) [SE]	Develop a routine (S4), block time for tasks in calendar (S2), ask co-workers for help (S27), ask specific questions (S28)	monitor workflow improvements
Improve myself (7)	Reflect on progress towards goals/tasks (S8)	self-reflection

The numbers in parentheses are counts for developers that set one or multiple goals for the goal category. Goals marked with [SE] are specific to software engineering.

7.5 Potential Impact of Reflective Goal-Setting (RQ3)

To learn more about the potential impact of reflective goal-setting on developers' goals and strategies and answer **RQ3**, we compared participants' responses describing their goal-setting practice prior to the study with their feedback after our study.

7.5.1 Self-Reflections can Help to Identify Concrete Goals and Actionable Strategies

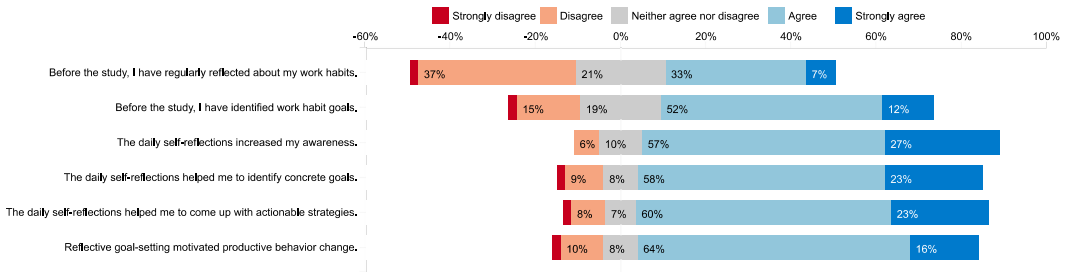
In our review of related work, we learned that while developers are interested in setting goals to maintain good work habits, it can be challenging to do so. Hence, we developed a reflective goal-setting study that allowed participants to self-reflect on work *purposefully*, and thereby encourages Hawthorne-type effects, by encouraging participants to alter their behavior based on the insights they gained from participating in the study. The result of participants self-reflecting and reviewing their work on a daily basis was that they started to validate their own experiences at work, and experiment with ideas for self-improvements. 84.5% of the 52 participants reported that reflecting on the progress they had made towards planned achievements and positive or negative aspects of the workday, raised their awareness about their existing work habits and progress at work (see Figure 7.1 for details):

> "*The daily self-reflection process was very interesting and the most informative for me since it gave immediate feedback about how I did during the day.*" - P6

> "*I found the [achievement] setting and subsequent checking of progress very valuable. Especially the parts where I planned to do X, Y and Z and ended up not doing any of them.*" - P33

After a few days of getting used to reflecting about work on a daily basis, participants started to **identify concrete goals**, which 80.8% of the 52 participants attributed to the *constant* self-reflections (Figure 7.1):

Figure 7.1: Participants' Self-Reports on the Value and Impact of Self-Reflection



> "*It was a wonderful exercise. It was not at all painful to start tracking (which I initially thought it would). Also, without self-reflection, I was not really seeing where I was spending much of my time and in-turn not able to correct/fix the challenges.*" - P48

> "*[The] study itself was almost a tool!*" - P45

On subsequent days, the self-reflections allowed participants to refine these goals or identify other opportunities for self-improvement that they transformed into new goals. These goals and the daily self-reflections allowed 83.3% of the 52 participants to experiment with and **develop actionable strategies** to reach their goals (Figure 7.1).

The study fostered participants to engage in deep reflection and self-awareness activities to better understand what hindered their ability to work productively and make progress, and also led to the identification of a wide variety of concrete work habit goals and actionable strategies. Overall, 30.7% (16 of the 52 participants) reported having identified new goals, 53.8% (28 participants) improved and refined existing goals, and only 15.5% (8 participants) reported that they did not identify any new goals, since they already had an elaborate goal-setting system in place (5 participants), or were happy with their current work habits (3 participants). We thus conclude that ***continuous* reflective goal-setting can be an important step towards improving and maintaining good work habits**.

7.5.2 Self-Reflection can Increase Awareness on Goal Achievement and Productive Habits

Besides facilitating goal and strategy identification, participants reported that reflective goal-setting increased their awareness on goal-achievement and productive work habits. The daily self-reflections allowed participants to evaluate the progress they made towards their goals and refine them when necessary, effectively providing a feedback mechanism on goal-achievement which previously was shown to be an important aspect of lasting behavior change [Humphrey, 1996; Kersten-van Dijk et al., 2017; Sharot et al., 2011]. Besides, the self-reflections increased participants' awareness about productive work habits, such as the importance of planning, of reducing interruptions and multi-tasking, and of clarifying problems before starting to work on them:

> "*I had long ago forgotten the utility of a short-term TODO list - something focused on just the next 24 hours. By starting to use a daily list, I could get a tight feedback loop between my plans and my actual outcomes.*" - P33

> "*I can now make this conscious decision: When someone is coming by I can decide 'yes I have time' or 'no I am going to focus right now'. Whereas before, I was like 'okay, I am here, let's just talk to them'.*" - P3

As a result, many participants stated that the motivation to finish the study was rarely about actually reaching their goals, but about the progress they made towards them and the **productive habit changes and learnings goals entail**:

> "*Defining goals does not have to be to achieve something. Making progress is a good goal as well.*" - P16

7.5.3 Reflective Goal-Setting can Increase Productivity and Well-Being

40 of the 50 participants (80.0%) who answered the question, agreed that the continuous self-reflections and goal-setting led to positive behavior change (Figure 7.1). We refrain from repeating the self-reported behavior changes here,

since they are congruent with the identified goals and strategies presented above. 12.0% (6 of the 50) who stated that they did not change their behavior, explained that they either already had good habits in place (3 participants), had difficulties with breaking bad habits (2 participants), or were not able to identify good enough opportunities for self-improvements (1 participant).

37 of the 40 participants (92.5%) who self-reported that they changed their behavior stated that they plan to keep their new behavior, mostly because they were satisfied with their **increased progress and (perceived) productivity**. **Higher well-being** was another benefit of the self-reflections, since they allowed participants to better detach from work (similar to [Williams et al., 2018]):

> "*It was a very good opportunity to understand more about myself. When I used to write it everyday, I realised that I am dissatisfied with myself even though I work to the best of my ability most of the time. And instead of getting dissatisfied, I should think of what I can do better the next day, and enjoy the rest of my evening with peace. Also, I learnt to think about my work in a more organised way. That will really help me in improving my day to day productivity and hence my happiness.*" - P32

Note that the duration of our study does *not* allow to demonstrate any long-term behavior changes, but it shows the efficacy of our reflective goal-setting approach to identify concrete and actionable behavior change plans, which should be what is evaluated in early stage personal informatics systems according to Klasnja et al. [2011].

7.5.4 Help Developers to Help Themselves

While our reflective goal-setting study was initially met with quite a bit of skepticism, since we "forced" (P8) participants to "write a diary" about their work, many reported that after a few days of **getting used to and practicing self-reflecting**, it became easier to self-reflect on their work and progress, which led to many surprising insights into their own work habits, and the identification of meaningful goals and strategies. Surprisingly, the approach was also valuable for participants who declared that they previously engaged in regular self-reflection and/or goal setting (Figure 7.1): Of the 63.5% (33 out of 52) who reported they

previously set goals for themselves, 84.8% (28 out of 33) identified new or revised existing goals; and of the 40.4% (21 out of 52) who regularly self-reflected prior to the study, 66.7% (14 out of 21) identified new or revised existing goals in a meaningful way. In the final survey, the majority of participants stated that they want to **continue doing self-reflections voluntarily** after the study; 40.4% daily (21 out of 52), 11.5% every two to three days (6 out of 52), and 44.2% weekly (23 out of 52). Several participants stated that they enjoyed participating in the study:

> "*Thank you for enforcing me to reflect daily.*" - P8

Only 4 out of 52 participants (7.7%) reported that they were happy the study was over and they would not want to continue doing self-reflections, either because the self-reflection caused anxiety (1 participant) or they were burdensome and tedious (3 participants), especially when performed daily:

> "*It was tedious. I would not like to do it everyday. I feel like the to do list and how much did you finish each of them is enough, tracking the tasks that are still incomplete.*" - P34

We conclude that in most cases, encouraging participants to self-reflect regularly can lead them to recognize its value, and the desire to develop a self-reflection routine to foster productive self-improvements.

7.6 Summary of Results

Our analysis provides insights into developers' goals to improve and maintain good work habits, and the value and impact of reflective goal-setting in the workplace. Table 7.4 summarizes our key findings.

Table 7.4: Summary of the Study Key Findings.

#	Finding	Section
F1	Developers' work habit goals can be categorized into improving time management, avoiding deviations from planned work, improving impact on the team, maintaining work-life balance and continuous learning.	7.4
F2	Most work habit goals describe *continuous* behaviors they want to develop, rather than momentary goals with a defined outcome or result.	7.4
F3	Continuous self-reflection can be an important step towards productive self-improvements at work, by supporting the identification and monitoring of work habit goals and strategies.	7.5.1, 7.5.2
F4	Reflective goal-setting can increase productivity and well-being at work.	7.5.3
F5	After developers are encouraged to self-reflect daily for a while, many recognize the value of self-reflection and want to develop a self-reflection routine.	7.5.4

7.7 Discussion

Our work provides insights into developers' work habits they consider desirable to pursue and set as goals for self-improvements. The majority of developers' goals describe *continuous* work habits, rather than momentary goals that have a clear outcome or result, suggesting that developers' motivation is less about actually reaching their goals, but about the progress they make towards them and the productive habit changes they entail. Recent work by Clear [2018]

supports this finding, stating that people should focus on changing their habits, rather than working towards a goal, which upon completion becomes irrelevant and causes many to relapse to old or previous habits.

While developers generally had individual differences in their goals and strategies to improve their work, we observed that the overarching objective was to **gain back control of their work to make more progress** on their tasks. Extensive studies performed by Amabile and Kramer [2011] support the observation and showed that the single best motivator at work is to empower knowledge workers to make progress, which also increases their happiness. Similarly, recent work suggested that many developers want to **gain back control over how they spend their** time at work, since work is often very randomized and fragmented by interruptions and meetings [Mark et al., 2018; Meyer et al., 2019]. The remainder of this section discusses how self-reflection and various tool support might help developers retain control over their work, make more progress and increase productivity.

Self-Reflection as the Missing Key to Productivity? Our study showed that reflective goal-setting provides developers with a framework for identifying goals and strategies that are *relevant* to their work, and motivates productive behavior changes. These findings are congruent with previous work that demonstrated the value of self-reflection for identifying and reaching goals in several areas of life, especially health, sleep and physical activity (*e.g.*, [Gasser et al., 2006; Kocielnik et al., 2018; Lee et al., 2017; Williams et al., 2018]).

While it is common for developers to review code, progress (*e.g.*, in scrum meetings) and performance (*e.g.*, reviews with managers), we are curious to learn in the future why they rarely reflect on work habits in practice, given the great promise. One reason could be time pressure, which makes it difficult to step away from work and take the time to reflect [Di Stefano et al., 2016]. Previous work has also suggested that self-reflection does not feel natural to some, which is why they need to be encouraged to try it and learn about the benefits from self-reflecting regularly themselves, similar to our own observations [Gustafson and Bennett Jr, 2002; Moon, 2013]. As noted above, many developers were

initially skeptical to participate in a self-reflection study, but *encouraging* them to try it out for a few weeks let them realize the value and leads to the desire of self-reflecting also outside of the study context. While we did not investigate the reasons for their initial skepticism, future work could study barriers towards initiating and maintaining continuous self-reflection, to better understand how we can encourage and convince developers. Our study did not look into long-term engagement with reflective goal-setting, but we assume that reflecting daily might be perceived as too cumbersome or time-consuming after a while. Varying with the intervals and frequency at which developers self-reflect might be one way to find a **trade-off between the cost and value of self-reflection** and motivate long-term engagement. For example, developers could initially reflect on a daily basis for a few weeks, to practice self-reflection and learn about the value, and then develop a habit of reflecting once or twice a week, or even only a few times a month. Participants further suggested various opportunities to reduce the cost of self-reflecting by better incorporating it into their daily work lives. These suggestions that participants brought up in the final survey are discussed in relation to previous work in the area of personal self-improvement and productivity in the remainder of this section.

Supporting Goal-Identification. To support goal-identification, participants suggested that self-reflection should be integrated into existing systems and workflows. Instead of creating another separate tool, recent work has explored how to integrate self-reflection into existing communication tools, such as Slack or Skype by building conversational bots [Kocielnik et al., 2018; Williams et al., 2018]. Furthermore, our participants and previous work [Baumer et al., 2014; Krogstie et al., 2012] emphasized that supporting self-reflection with computerized systems is crucial. For example, through automated monitoring, which can provide personalized insights and statistics into developers' work, and ease the recollection part of the reflection and foster goal-identification. Existing automated monitoring systems successfully increased knowledge workers' awareness about specific aspects of work, but the provided insights were often not actionable enough for users to know how and what to change, which is why the engagement

with these tools usually decreased after a few days [Collins et al., 2014; Kim et al., 2016; Li et al., 2010; Meyer et al., 2017b]. According to Baumer et al. [2014], the problem of most automated monitoring research is the implicit assumption that just by providing access to "prepared, combined, and transformed" data, in-depth reflection can and will occur. We are, thus, interested in studying how automated monitoring can be *combined* with self-reflection, to reduce the time and effort required to participate in active, in-depth reflection, while still providing rich and actionable insights. For example, the automated monitoring could provide developers with some automatically generated insights and visualizations on how they spent their time at work, and then still prompt developers to actively self-reflect about their workday.

Since self-reflection is typically reactive [Johnson and White, 1971], participants suggested that receiving examples and recommendations for self-improvements, that are based on their current behaviors and compared to best practices or developers with similar job profiles, could be valuable for a proactive goal-identification. Such a recommendation system could, for example suggest good moments to take a break when a developer is stuck on an issue [Epstein et al., 2016a]. However, recommending relevant goals to developers is challenging, since there might be privacy concerns from collecting the data, and since what is a good work habit to one developer is not necessarily a good one for another. For example, one developer might want to reduce interruptions from co-workers to have more uninterrupted time to focus on coding tasks, while another developer finds value and satisfaction in the resulting discussions. The advantage of self-reflections is that they allow developers to personalize and tailor work habit goals to their needs, with a minimum of privacy concerns. Recent work on improving sleep, physical activity or living with diabetes has shown early promise of building virtual "coaches" or "assistants" that provide personalized and tailored recommendations, *e.g.*, in the form of conversational bots [Daskalova et al., 2016; Herrmanny et al., 2016; Monkaresi et al., 2013]. However, many open questions remain as to how to best tailor these systems to users, *e.g.*, by altering the timing and content of their recommendations.

Monitoring Goal-Achievement. Reflective goal-setting allows developers to gain self-generated feedback on their goal-achievement, a key pillar of successful behavior change [Humphrey, 1996; Kersten-van Dijk et al., 2017; Sharot et al., 2011]. To further improve the trade-off between cost and value of self-reflections, participants suggested how the progress towards their goals could be automatically measured, instead of having to manually self-report them each day. These suggestions include the monitoring of switches between activities and tasks and updates to the task list, WI tracker, calendar and documentation; they are summarized in the last column of Table 7.3. Other goals, such as if a developer became a better colleague, are more challenging to track automatically.

Supporting Goal-Maintenance. Finally, participants described how tools could support the maintenance of goals once they reached them (*i.e.*, maintaining good habits), either passively or actively. One less intrusive way that participants suggested is to show reminders that help to avoid forgetting to pursue their goals. Previous work suggested that reminders need to be context sensitive to be effective, and that nudging users into performing a behavior might be even more successful [Fogg, 2003; Munson and Consolvo, 2012; Prestwich et al., 2009]. Similarly, other participants asked for a system that automatically interferes with their work if needed. For example, by hiding email or instant messaging notifications at times of high focus, by blocking distracting websites, or by disallowing participants to switch to another task when they committed to work on a specific one. However, participants were fairly ambivalent about the value and risks of such systems, suggesting they would require a fair amount of customization and contextual awareness to work well. The finding is supported by recent work by Mark *et al.* who found that blocking online distractions increases focus and productivity, but at the cost of higher stress [Mark et al., 2018].

Related research in the physical activity domain has shown promise in motivating people to maintain their goals by including social challenges and competitions (*e.g.*, [Edelson et al., 2011; Fogg, 2003; Fritz et al., 2014; Lin et al., 2006; Rooksby et al., 2014]). For example, Chick Clique, a system that allowed teenage girls to share their health and phyiscal activity related data, showed that data sharing can be a powerful motivator [Toscos et al., 2006]. However, when we asked our

participants about comparing themselves to or competing with co-workers, 62.7% (32 of 51) stated clear disinterest. Reasons were either that work is too individual to be quantified and compared or that some people would start to "game" these metrics:

> "*Everyone works in different ways and such comparative measurements seem to generally fail to capture that to any satisfactory level. Additionally any such comparisons are likely to be gamed by the people who value them the most resulting in workers who strive to increase their metrics rather than being effective and creative.*" - P01

According to Treude et al. [2015] and Muller [2018], users are most likely to "game" a system to receive higher scores when the measure seems inaccurate or unfair. Furthermore, research on including social components with physical activity trackers to motivate self-improvements showed similar privacy concerns, increased pressure, and reported that participants felt "awkward" when sharing data with strangers [Fritz et al., 2014; Lin et al., 2006; Prasad et al., 2012; Toscos et al., 2006]. One way to overcome these challenges could be to develop support groups—an inner circle of people who want to encourage a person to reach self-improvement goals—but it was shown they are challenging and time consuming to build and maintain [Munson and Consolvo, 2012].

7.8 Threats to Validity

External Validity. Although we studied 52 software developers at 10 different companies (startups to multi-national companies) from four countries, generalizability to other development companies, to other developers, and other knowledge workers, might be limited. The positive effect of self-reflection on goal-identification and -achievement might be threatened by a self-selection bias of participants who are generally more interested in better understanding their work and improving their practices, which is a common threat in self-improvement research [Kersten-van Dijk et al., 2017]. After a few days of skepticism, most participants started to appreciate the value of self-reflecting and goal-setting, but it is unclear how well this generalizes to other developers. We tried to

mitigate the self-selection risk by being very upfront about the study objectives and method, by stressing that the collected data is and will remain private, and by allowing participants to continue their regular work and selecting a suitable time for participation.

Internal Validity. While we relied on methods and findings that were successfully applied in previous work in other fields (see Section 7.3), the design of the reflective goal-setting study and framing of the daily self-reflection questions might have influenced participants. For example, since we prompted participants to reflect on progress and achievements, the goals they identified might have been biased more towards these. Furthermore, the prompts for daily self-reflection and goal-setting in our study influenced participants' behaviors (Hawthorne effect), but this was intended to answer RQ3. While most identified goals overlapped with several other participants, another threat to the internal validity of our results is based on our reliance on participants' self-reports only. For example, the strategies we identified to successfully support goal-achievement rely on participants' accurate and non-exaggerated reporting. To mitigate the risk, we first carefully analyzed each participants' self-reports individually, before comparing them to their study feedback. In case their responses showed decreasing interest (*e.g.*, missing multiple days of self-reports) or the time spent per self-report was very short, we discarded their data. In addition, the positivity towards self-reflection might be caused by novelty effects, and might wear off after participating for multiple months. In Section 7.7, we discuss how tool support might motivate long-term engagement and help to balance the trade-off between cost and value of self-reflections.

Construct Validity. We performed a Thematic Analysis [Braun and Clarke, 2006] to analyze participants' self-reports and responses to the pre-study and final surveys. One potential threat could be that the open coding step was performed by one author only. To reduce bias, we discussed themes and quotes of each coded category in the team.

7.9 Conclusion

In this paper, we explored software developers' goals and strategies to improve or maintain good habits at work. We identified five main goal categories, and found that developers generally want to develop good continuous behaviors, rather than reaching momentary goals with a defined outcome and result. We show that *continuous* self-reflection can increase developers' awareness about work habits, and that it can also lead to productive behavior changes that increase productivity and well-being at work. Our results suggest that *purposeful* and active self-reflection can provide actionable insights into potential self-improvements at work, something which has previously been shown to be challenging with existing self-monitoring approaches in the workplace. We discuss approaches to find a trade-off between the cost and value of workplace self-reflection, and how tools could potentially support goal-identification, goal-achievement monitoring, and support the maintenance of goals and good habits.

7.10 Acknowledgements

We thank our study participants for their participation. We also thank the anonymous reviewers and our editor for their valuable feedback.

8

Reducing Interruptions at Work: A Large-Scale Field Study of FlowLight

Manuela Züger, Christopher Corley, ***André N. Meyer****, Boyang Li, Thomas Fritz, David Shepherd, Vinay Augustine, Patrick Francis, Nicholas Kraft, Will Snipes*

Published at the 2017 CHI Conference on Human Factors in Computing Systems

Contribution: Involvement in all main parts of this large team project, including the tool development, execution of field study, data analysis, and paper writing. We licensed the FlowLight in 2018 to be commercially sold by Embrava [1].

[1] `https://embrava.com/pages/flow`

Abstract

Due to the high number and cost of interruptions at work, several approaches have been suggested to reduce this cost for knowledge workers. These approaches predominantly focus either on a manual and physical indicator, such as headphones or a closed office door, or on the automatic measure of a worker's interruptibility in combination with a computer-based indicator. Little is known about the combination of a physical indicator with an automatic interruptibility measure and its long-term impact in the workplace. In our research, we developed the FlowLight, that combines a physical traffic-light like LED with an automatic interruptibility measure based on computer interaction data. In a large-scale and long-term field study with 449 participants from 12 countries, we found, amongst other results, that the FlowLight reduced the interruptions of participants by 46%, increased their awareness on the potential disruptiveness of interruptions and most participants never stopped using it.

8.1 Introduction

Knowledge workers are frequently interrupted by their co-workers [Czerwinski et al., 2004; González and Mark, 2004; Sykes, 2011]. While many of these interruptions can be beneficial, for instance to resolve problems quickly [Isaacs et al., 1997], they can also incur a high cost on knowledge workers, especially if they happen at inopportune moments and cannot be postponed [Bailey and Konstan, 2006; Borst et al., 2015; Mark et al., 2008b; McFarlane, 2002].

Due to the high cost and the high number of interruptions that knowledge workers experience every day (*e.g.*, [Czerwinski et al., 2004; González and Mark, 2004]), several approaches have been proposed that can roughly be categorized by the interruptions they address: computer-based and in-person. Studies have shown that the cost of computer-based interruptions can successfully be mitigated by automatically detecting a knowledge worker's interruptibility and mediating interruptions by deferring them to more opportune moments (aka. defer-to-breakpoint strategy) [Arroyo and Selker, 2011; Ho and Intille, 2005;

Iqbal and Bailey, 2008]. Another strategy to reduce the cost of computer-based interruptions is to indicate a person's interruptibility to co-workers in a contact-list style application on the computer [Begole et al., 2004; Lai et al., 2003; Tang et al., 2001]. While these approaches have also been suggested for addressing in-person interruptions, they did not show to have any effect on them, probably since the contact-list style applications can easily be hidden behind other applications and thus forgotten at communication initiation [Begole et al., 2004; Fogarty et al., 2004; Hincapié-Ramos et al., 2011a; Lai et al., 2003].

For in-person interruptions—one of the most costly kind of interruptions due to their high frequency and immediate nature [González and Mark, 2004; McFarlane, 2002; Sykes, 2011]—approaches predominantly rely on manual strategies to physically indicate interruptibility, such as wearing headphones, closing the office door, or using busy lights that have to be set manually [Blynclight, 2019; Sykes, 2011]. Since manual approaches are cumbersome to maintain, users generally don't update them on a regular basis and their accuracy and benefits are limited [Milewski and Smith, 2000]. Only very few approaches have looked at a combination of a physical interruptibility indicator with an automatic interruptibility measure to reduce the cost of in-person interruptions [Bjelica et al., 2011; Hincapié-Ramos et al., 2011b] and there is no knowledge on the long-term effects of such approaches.

In our research, we developed the FlowLight approach, an approach to reduce the cost of in-person interruptions by combining a physical interruptibility indicator in the form of a traffic-light like LED (light emitting diode) with an automatic interruptibility measurement based on a user's computer interaction. In a large-scale and long-term field study with 449 knowledge workers from 12 countries and 15 sites of a multinational corporation, we evaluated the FlowLight and its effects in the workplace. Over the course of the study, we collected a rich set of quantitative and qualitative data, including self-reported interruption logs of 36 participants, survey responses of 183 participants that used the FlowLight for at least 4 weeks, and in-depth interviews of 23 participants. Our analysis of the data shows, amongst other results, that the FlowLight significantly reduced the number of interruptions of participants by 46%, while having little impact on

important interruptions. Further, the FlowLight increased the awareness on the cost of interruptions within the workplace, participants felt more productive using the FlowLight and most participants continued using the light for up to 13 months by now. Overall, the gained insights on the long-term usage of the FlowLight provide strong support for the benefits of combining a physical interruptibility indicator with an automatic interruptibility measure in the workplace and its significant impact on reducing in-person interruption costs.

8.2 Related Work

Related work on managing interruptions can broadly be grouped into strategies for reducing interruptions and disruptiveness, and ways of measuring and indicating interruptibility. To avoid repetitions in this thesis, we present and summarize the related work in Section 1.6 of the synopsis. Related work on measuring interruptibility are specific to this publication and are presented below.

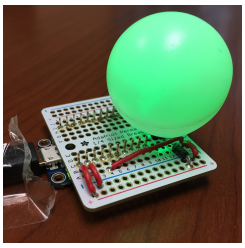
8.2.1 Measuring Interruptibility

Previous research has explored various features to measure a person's interruptibility. For instance, Hudson et al. [2003] simulated sensors by coding audio and video recordings into features related to the person's current context, such as the number of people present or the phone being on the hook. While their approach showed promise in measuring interruptibility, the chosen features are difficult to capture automatically.

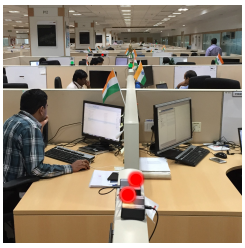
To automatically detect a person's interruptibility, Stern et al. [2011] developed an approach that is based on the person's location and calendar information. Fogarty et al. [2005] used speech sensors, location and calendar information and activity on the computer to measure presence and availability; Tani and Yamada [2013] measured interruptibility using the pressure applied on the keyboard and mouse; and Coordinate by Horvitz et al. [2002] uses user activity and proximity of multiple devices to forecast presence and availability.

More recently researchers have also started to use biometric data to measure

Figure 8.1: Evolution of the Physical Indicator of the FlowLight Over Time



(a) July 2015: Original Prototype



(b) August 2015: First Major Pilot (Lights Emphasized with Overlays)

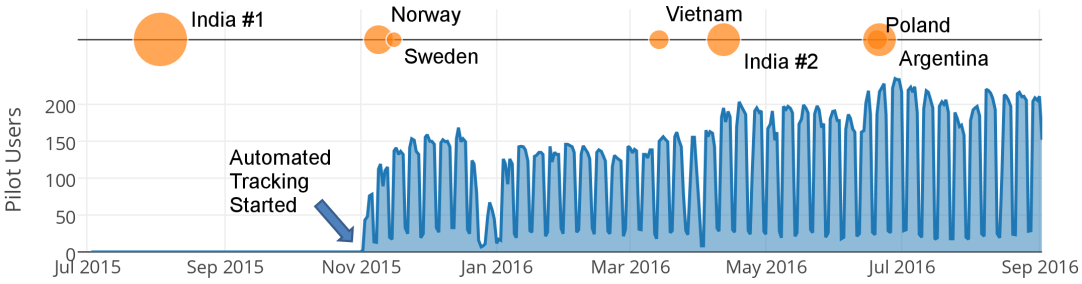
(c) October 2015: Blink(1) Version with Adhesive Clip

(d) April 2016: Deployment in Second Pilot in India (India #2)

interruptibility. For instance, Mathan et al. [2007] classified interruptibility during a US military training with an electroencephalography (EEG) sensor that captures the electrical activity of the brain. Chen et al. [2007] calculated interruptibility based on an electromyography (EMG) sensor that captures heart rate variability and muscle activity. In our previous work, we used various biometric sensors (EEG, electrodermal activity (EDA), skin temperature, and photoplethysmography (PPG)) to predict interruptibility [Züger and Fritz, 2015]. Overall, research has shown that biometric sensors can be valuable in automatically measuring interruptibility, however, at this point the biometric sensors required to accurately measure interruptibility are generally still too invasive for long-term usage.

The FlowLight builds upon previous research in this area by automatically measuring interruptibility based on a combination of computer activity, calendar information and log-in state. It thereby utilizes a minimally invasive set of features that performs well without compromising the users' privacy or requiring additional body-worn biometric sensors. It further extends previous research in this area by combining the automatic measure with a physical indicator.

Figure 8.2: FlowLight Users Over Time (Size of Orange Circles Indicates the Number of Participants; Regular Dips in the Number of Pilot Users Represent Weekends and the Prolonged Dip in December/January 2016 represents the Christmas Break)



8.3 Approach and Implementation

The FlowLight consists of a computer application to automatically determine a user's interruptibility state and a physical LED light to indicate this state to co-workers. The FlowLight was developed iteratively over more than a year and improved continuously based on feedback from a small developer team that we used for testing, and later on, also based on feedback from study participants.

Physical LED Light. FlowLight uses a physical traffic-light like LED to indicate the interruptibility status to co-workers. This light has evolved throughout the pilots[2]. The first model, which was designed and soldered in-house, is shown in Figure 8.1a. In Figure 8.1b the same model light is shown encased in plastic and deployed in an open office space. Finally, Figure 8.1c shows the blink(1)[3] LED light that we adopted to avoid installation issues with certain drivers immediately after the first major pilot, which was also the first of two pilots in India (denoted as India #1 in Figure 8.2). Typically, we mounted the LED light on a user's cubical wall or outside a user's office.

The light uses different colors to indicate four states: *Available* as green, *Busy*

[2]We use the term *pilot* to refer to each individual field study trial with a separate team.
[3]`https://blink1.thingm.com/`

as red, *Do Not Disturb (DnD)* as pulsating red, and *Away* as yellow. Note that these states and colors mimic the ones used by prominent instant messaging services, in particular the one used by the company under study.

Application. The application features three main components: a *Tracker* to capture events relevant for calculating the interruptibility state, a *Status Analyzer* to analyze the captured events and calculate the user's interruptibility state on the fly, and a *Status Manager* to manage the user's current status, propagating it to the LED light and other applications, in particular instant messaging (IM) clients. The application was implemented to be compatible with the Windows operating system, Skype for Business, an IM and video-conferencing system, and Office 365, a software suite that provides email and calendaring services, amongst others. We chose to tailor our application to these systems and applications due to the IT setup at the target company for our study.

The *Tracker* logs a user's mouse and keyboard interaction. In particular, it collects mouse clicks, movements as pixels moved, scrolling as pixels scrolled and keystrokes (without recording the specific key). This component also logs calendar events to determine meetings and the Skype status.

The *Status Analyzer* uses the tracked keyboard and mouse events to calculate the user's interruptibility status on the fly, i.e., whether the user is available, busy, highly busy (DnD) or away. The algorithms used to calculate the interruptibility status are described below.

The *Status Manager* is notified by the Status Analyzer at every change in the user's interruptibility, and then propagates the updated status to the physical LED light and the user's presence status in Skype for Business. The presence status in Skype for Business can also be changed manually by the user, or automatically by the Office 365 calendar, in case a meeting is scheduled. In case the presence status is changed manually, the Status Manager updates the interruptibility state of the application and the physical LED light.

Algorithms for Status Updates. Over the course of this study, we used three different algorithms to determine and update the interruptibility status automat-

ically, improving them based on critical user feedback as discussed below.

FlowTracker. This algorithm sums up the computer interaction in the past three minutes according to heuristic weights assigned to each type of event, which were tuned based on feedback from early alpha and beta users of the FlowLight. If the value of the sum is in between the top 9% and the top 4% of their activity range—we captured averages over the past days—the user is considered busy. If it is within the top 4%, the user is considered highly busy. In our first pilot study in Bangalore, India (India #1 in Figure 8.2), we used different thresholds at first, namely 13% and 5% based on a prior study that indicated that knowledge workers are not interruptible for approximately 18% of their day. However, several technical writers (and others) involved in that pilot gave strong feedback that the light switched to the busy state too easily, which is why we lowered the thresholds to the mentioned 9% and 4%.

Smoothing. While the FlowTracker showed promise, many early users complained that it was too sensitive to certain input. For instance, a twenty second burst of typing may cause a user to temporarily be shown as busy. Therefore, the Smoothing algorithm marks users as busy if they were active in each of the last three minutes and exceeded a threshold of 100 combined mouse clicks and key presses in the recent past (between 4 and 7 minutes ago). This algorithm reduces frequent changes by requiring over three minutes of activity to become busy and, once busy, by requiring only one above-threshold minute in the recent past to remain busy. To achieve the highly busy status, users had to be busy at the current point in time and had to be above-threshold for fifteen of the last thirty minutes.

Smoothed FlowTracker. While the Smoothing algorithm leads to fewer status changes, since it relied on a static threshold (i.e., 100 combined mouse clicks and key presses), it did not adapt to individual users' work patterns. For instance, designers working on drawings tended to use mouse clicks almost exclusively, which makes it difficult to exceed the threshold. Thus, we finally combined the FlowTracker algorithm with the Smoothing algorithm to achieve the advantages of both approaches. This algorithm, currently in use, operates as the Smoothing algorithm, but instead of using a static threshold, it utilizes the FlowTracker

algorithm to determine above threshold values. This algorithm eliminated all of the most common complaints reported by pilot users. Further refinement of the algorithm is left for future work.

Although our main intent was to use an algorithm to infer interruptibility, we offered participants a "Manual Only" mode since it was requested by some participants, especially those with management roles that needed to be available to others most of the time, and we noticed (and our study confirmed) that our algorithms might not be accurate for everyone or for all activities requiring focus, such as reading or thinking.

8.4 Evaluation

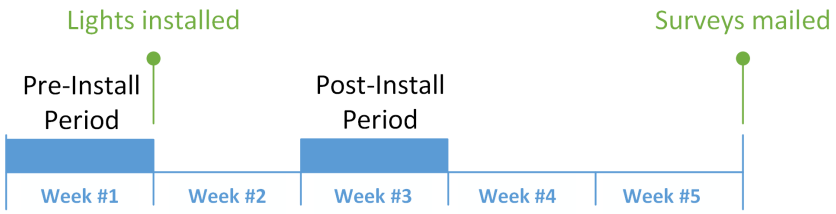
To evaluate the FlowLight, in particular the combination of the physical indicator and the automatic interruptibility measure as well as its effect on knowledge workers, we conducted a long-term and large-scale field study with 449 knowledge workers. For this study, we installed the FlowLight at over 15 locations in 12 countries of one multinational corporation. Over the course of the study, we collected a rich set of data using a combination of experience sampling, a survey, an interview and computer interaction monitoring. Figure 8.1 illustrates a few pictures of the FlowLight in use in different pilots. Figure 8.2 indicates the increasing and continuous number of participants and the major pilots of this study since its beginning and up to September 2016.

8.4.1 Study Procedure

For each team participating in our field study we conducted the same five-week pilot procedure as illustrated in Figure 8.3. Prior to the start of a pilot, we asked the participants to install the FlowLight application in 'data collection only'-mode.

In *Week #1* of the pilot, users were instructed to use the FlowLight application to manually log the time and severity of each interruption during the five day work week. Our application allowed participants to log interruptions by a click

Figure 8.3: Timeline of Study Procedure



on the taskbar menu or a hotkey combination for minimal invasiveness. As soon as an interruption had been logged, a single modal dialog appeared that asked participants to specify the severity of the interruption on a 5-point Likert scale.

At the beginning of *Week #2*, the physical indicator of the FlowLight was installed and the automatic status update feature for the interruptibility status was activated. To minimize Hawthorne-type effects and have participants and co-workers get used to the FlowLight, we then waited for one week before we gave further instructions.

At the beginning of *Week #3*, we again asked participants to manually log their interruptions for 5 work days. We also reminded participants about the manual logging in Week #1 and #3 to ensure they would not forget.

During *Week #4 and #5* users continued using the FlowLight. Throughout these 5 weeks the application collected anonymized usage data. At the end of *Week #5*, after participants had the FlowLight for four weeks, our application prompted them to complete a survey. The survey took an average of 14.2 minutes to complete and had questions on the FlowLight approach and its impact, in particular on participants' interest in continuing using the approach, its impact on interruption costs, productivity and interaction behavior, on the accuracy of the automatic state detection and manual setting, as well as on general feedback and demographics. After completing the survey, users were asked on the last page of the survey to upload their data collected by the FlowLight application, which included the usage data logs and the logs of the manually captured interruptions.

For a deeper understanding of the long-term usage, experience and effect of the FlowLight, we conducted in-depth interviews with a subset of participants approximately two months after they installed the FlowLight. Interview par-

ticipants were selected semi-randomly, based on accessibility, availability and willingness to participate in the interview. The interviews were on average 19.5 minutes long and the questions focused on the benefits and limitations participants observed with the FlowLight approach, as well as on how it impacted their own behavior and interactions in the team over the course of the two months since the installation. For instance, we asked participants whether they felt that their colleagues respected their FlowLight or if they noticed situations in which the status was not accurate. Note that the interview and survey questions can be found on our supplemental materials site [4].

Independent of the timeline of the study procedure, we also started to anonymously log the number of people running the FlowLight application each day. For privacy reasons, we only keep track of the number of unique active FlowLight users in the online log.

8.4.2 Participants

Since the beginning of our study 13 months ago, we installed the FlowLight approach with a total of 449 participants from 15 sites, located in 12 different countries, of one multinational corporation. From these 449 participants, we were able to gather:

Survey responses from 183 participants (IDs: S1-S183), 144 male and 39 female, with an average age of 36.0 years (standard deviation, in the following denoted with ±, of 8.7), an average professional experience of 12.0 years (± 8.0), from a variety of work areas, including 77 participants in development, 56 in other engineering, 24 in project management, 15 in other non-engineering, and 11 in testing, and with various job roles, including 70 individual contributors, 36 other, 32 leads, 31 managers, 8 executives, and 6 architects;

Interview transcripts (conducted by us) from 23 participants (IDs: I1-I23), 22 of which were male, 1 female, average age of 36.9 years (± 5.8), average experience of 13.2 years (± 4.7), and with various job roles, including 9 managers, 11 software developers, 1 researcher, 1 product owner, and 1 tester;

[4] `https://sites.google.com/site/focuslightonline`

Interruption logs (self-reported) from 36 participants across six different countries, 13 from Argentina, 6 from Norway, 5 from Poland, 5 from Switzerland, 5 from Sweden, and 2 from the USA;

Usage data logs from 47 participants (IDs: D1-D47) 20 from Argentina, 18 from India, 4 from Poland, and 5 from Vietnam.

Online logs from all 449 participants that installed the approach (each one had the application running for at least one day after we integrated the logging feature).

Note that due to privacy concerns with the collected data, we did not require participants to identify themselves in each step and/or fill in their demographics, except for the survey, which is why we can only report some demographics for each round and are not able to track the participants across the different methods, for instance the survey and the self-reported interruption logging.

8.4.3 Data Collection and Analysis

Survey and Interview. In total, we collected survey responses from 183 participants after they had been using the FlowLight for at least four weeks, and interview transcripts from the 23 participants after they had been using the FlowLight for approximately two months. To analyze the textual data of the survey and interview responses, we used techniques based on Grounded Theory, in particular open coding and axial coding to determine higher level themes. To establish a common set of codes and themes, two of the authors applied open axial coding to the same subset of interview transcripts and then established a common understanding and defined a structure for the most commonly mentioned concepts. As the topics of the survey and interviews overlap, we used and extended the same coding scheme to analyze the textual survey responses. To validate the analysis of the survey results, two additional authors extracted their main findings from a subset of the responses independently.

Interruption Logs. Interruption logs capture the self-reported interruptions per participant logged with the FlowLight software. We collected interruption logs

with at least two logged interruptions from 102 participants. We down-selected these to 36 logs by applying strict filtering criteria to ensure data validity as follows. We excluded all interruptions in all logs that were accidentally logged during the first five days after the installation of the FlowLight, as interruptions in the period right after the installation are not representative due to Hawthorne-type effects, such as participants getting used to the FlowLight, and co-workers asking curiosity questions. We then excluded all participants, that logged interruptions for fewer than three days in the pre- or the post-installation period. We chose three days as the threshold for each period to ensure a representative sample of work days for comparison without a too strong bias by individual outlier days. Each of the 36 interruption logs captured a combined average of 9.0 work days (± 2.2) for pre- and post-period, and contained an average of 28.9 total logged interruptions (± 17.0) per participant for the combined time period. We used these interruption logs to compare the impact of the FlowLight on the number of interruptions rated as disruptive by participants.

Usage Data Logs. We captured usage data logs from a total of 179 participants. These logs consist of computer interaction logs, such as mouse and keyboard events, and FlowLight usage data. Since we wanted to analyze user behavior before and after installing the light, we removed any logs that did not include at least two days before and after installing the light. We also excluded logs older than January 2016, as key usage messages were not yet logged by our software, making the analysis infeasible. We ended up with 47 usage data logs containing a total of 1560 work days. These logs consisted of an average of 7.3 work days (±4.2) prior to light installation and 25.9 work days (±14.0) after light installation per participant.

We analyzed usage logs in two ways. First, we counted the number of status change events recorded in the log per day per user for the period before and after the light installation event. It is worth noting that we only included usage logs within the five work days and not on weekends. Second, we used the intervals between status change events detected by one of our algorithms to determine how much time was spent in each status, again for before and after light installation.

To eliminate inappropriate intervals (*e.g.*,, a user did not turn off the workstation after work), we only accumulated the duration within 12 hours per day.

Online Logs. We collected online logs for a total of 305 days from November 2, 2015 until September 2, 2016 and from 449 participants. These logs were used to determine how many users were using the FlowLight on a given day (as shown in Figure 8.2). We analyzed these logs by summing up the number of unique identifiers that appeared in the log on a given day, which represents the number of active users for that day.

Based on participants' feedback during the period of the field study, we deployed the three main variations of the algorithm described earlier to set the status of the FlowLight. We analyzed differences between the data sets gathered with the three main variations of the algorithms and found no significant differences between the data collected with any two variations, neither in the collected survey items, nor the interruption logs. In the following, we will therefore present the results aggregated over all variations.

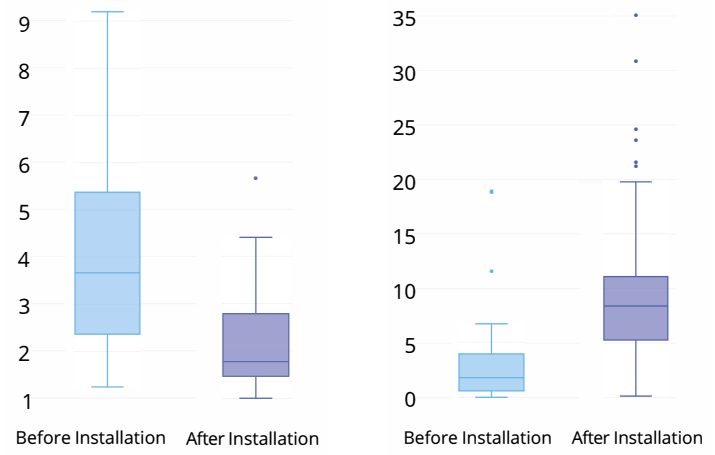
8.5 Results

In this section we present the primary findings of our field study. We first examine the effect of the FlowLight on the cost of interruptions before we examine how the FlowLight changed participants' interruption awareness, their interruption-related behavior, and their perception of productivity. Subsequently, we present insights on the costs of the approach, on the influence of its accuracy, on its continued usage by participants and on professional differences.

8.5.1 Reduced Cost of Interruptions

Figure 8.4a is based on the 36 collected interruption logs and illustrates the distribution of the number of interruptions per day and participant in the period before and the period after participants had been using the FlowLight for one week.

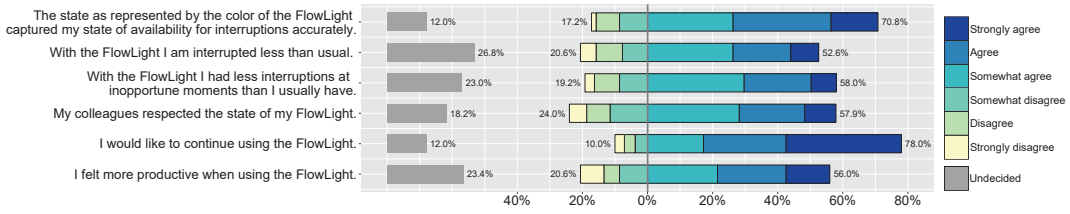
Figure 8.4: Logged Interruptions and State Changes Before and After Installing the FlowLight.



(a) Interruptions per Participant and Day.

(b) Number of State Changes per Participant and Day.

Figure 8.5: Results of a Subset of the Survey Questions.



Overall, the number of interruptions decreased after the installation and one week usage of the FlowLight by an average of 1.9 ($\pm$1.6) interruptions (46%) per participant and day, from 4.1 ($\pm$2.1) to 2.2 ($\pm$1.1). A Wilcoxon signed-rank test showed that this reduction is statistically significant (Z =-5.0, p <.000001).

A second Wilcoxon signed-rank test only on the number of severe interruptions (disruptiveness rating of 4 or 5) per day and participant further showed that there is also a statistically significant reduction with p <.001 and Z = -3.2.

An analysis of the survey results (see Figure 8.5 for more detail) further supports that installing the FlowLight reduced the cost of interruptions. 55.0%

of the 182 survey participants that answered the question stated that they either strongly agree, agree or agree somewhat that they were interrupted less than usual during their work, while only 20.3% disagreed with it. Even more participants, 59.3%, agreed that they had less interruptions at inopportune moments than usual, whereas only 19.8% disagreed with this statement.

During interviews participants echoed this quantitative evidence. In interview excerpts (full quotes listed in subsequent subsections) participants consistently mentioned that interruptions were reduced. They claimed that the pilot *"..resulted in less interruptions.." (S126)* , eliminated interruptions from colleagues (*e.g.*, *"When [the light]'s red I think they don't interrupt." (I11)*), and *"..didn't stop [interruptions] completely but they surely reduced." (S16)* .

Overall, our findings from the interruption logs, survey questions, and interview questions show strong support that the introduction of the FlowLight reduced the cost of interruptions in terms of the overall number as well as their severity.

8.5.2 Increased Awareness of Interruption Cost

After using the FlowLight for some time, participants developed a high degree of awareness for the cost of interruptions:

> "*It brings more awareness to what people are doing. Sometimes people take it for granted that people are always interruptible. But there is actually a cost or a penalty when you interrupt someone. So, I think just the concept is good because it reminds people that there is sometimes a good time and a bad time to interrupt people. So, I think just from an awareness campaign, it's valuable as well.*" - I20

> "*The pilot increased the sensitivity to interruption. Team members think more about whether an interrupt is necessary and try to find a suitable time.*" - S45

The FlowLight thereby served as a physical reminder for the interruptibility of co-workers in the moment and participants generally respect it and its state:

> "*It's kind of a like a mood indicator ... so it tells people the state ... of the owner of the light. And then it helps people be more aware or attentive to what my current situation is.*" - I18

> "*I think what really changed is ... a different consciousness about interruptions in our team and also with my colleagues ... I think ... they really respect the light. When it's red I think they don't interrupt.*" - I11

Overall, 70% of the 23 interview participants explicitly stated that the FlowLight is respected in their offices and 59.6% of 183 survey respondents agreed that colleagues respected the state of their FlowLight vs. 23.0% that did not (Figure 8.5).

The increased awareness and respect also triggered participants to change their behavior in a variety of ways, ranging from thinking twice before asking, to deferring the interruption, asking before interrupting and changing to a different communication channel, such as email or instant messaging:

> "*People ask each other if they are available, even when the light is green, even to people with no light. When I see the colleague I want to ask a question ... has a red light, then I wait a while, or write an email.*" - S77

> "*If it's red, I'll send them a message so that when they're no longer busy or something like that, they'll see the message and they can respond to it then ... so it doesn't require an immediate response*" - I19

Fortunately, participants used common sense when working with FlowLights. If a light was red or red blinking participants would still interrupt if the request was urgent:

> "*Once I go up there [to the person] and I see the light and then I also see that they're pretty intense then I'll push it off unless I really need to get answered to.*" - I17

8.5.3 Feeling of Increased Productivity and Self-Motivation

As a further effect of the FlowLight, 58.5% of the survey respondents felt more productive using it, while only 20.1% disagreed with this (Figure 8.5). This feeling of increased productivity often stemmed from the fewer interruptions:

> "*I definitely think it resulted in less interruptions both in person and via Skype. This resulted in more focus and ability to finish work.*" - S126

Another reason for the increased productivity is that the FlowLight serves

for some participants as a self-monitoring device that motivates them to become or stay focused, which, however, can also be distracting at times:

> "*Mostly it has helped as a personal monitor only for me. If I see the light red, I sense I am in the flow and I keep working.*" - I2

> "*When I notice that my light is turning yellow, and I'll feel like, 'Oh yeah, I've been idle' and then I do something ... I think the other way, yeah, there's some effect there too. Like, if I see that it's red, or even flashing red, then I'm like, 'Yeah, I've been very active, or productive, I should keep that going.' At the same time, I think it's also a little bit distracting too. Sometimes just because the light is there, I turn around to check it.*" - I12

8.5.4 Costs of Using the FlowLight

While people experienced reduced interruption costs and increase in productivity, there are also costs when starting to use the FlowLight. Especially right after installing it the curiosity of co-workers can lead to an increase of interruptions, which, however, diminishes after a few days:

> "*People walk by, they see it, they ask me questions, 'What's that? How does it work? What's going on?' like this.*" - I19

> "*Initially there were many people just curious to know what the light is about. This increased the number of interruptions but after few days, people started to respect [it].*" - S16

A few participants also experienced situations in which the FlowLight provoked interruptions, as the green color of the light might be misunderstood as an invitation (observed by 26% of the interviewed participants):

> "*What I definitely notice is that green is more inviting. So it actually encourages people to come by and say, "Hello" for me at least.*" - I20

In some cases, changing the interaction culture might require a mandate from higher up or can even be too expensive:

> "*The more important issue is for it to work, you have to have people committed to following the light rules, which probably requires engagement of some higher*

> *management ... and requires introducing the lights to a wider audience.*" - I6

> "*For us ... the main cost of introducing [it is] that you have to change how you are used to interact with people, that you first have to remember to take a look at the light. That's something that's probably too much for the team. [In] our environment .. it's easier to look at the people than at lights.*" - I8

If colleagues choose to ignore the light, especially for unimportant interruptions, it can lead to negative emotions:

> "*So, for us, what we also heard sometimes is that people have the light red, and others still interrupt them, and they're like, 'Oh no, I have this light red, why did they?' Like it bothers them, and it creates negative emotions almost more than it creates positive emotions...*" - I17

Finally, the public disclosure of the interruptibility status might make people feel exposed at times (8% of survey participants agree, 6% strongly agree) or lead to negative feelings:

> "*Oh, do other people see that my light is yellow? And are they thinking that I'm not working?*" - I12

Like any new technology, there is a cost to adopting the FlowLight. However, most of the identified costs diminish quickly or can be mitigated by clear direction from management. Overall though participants predominantly stated that the colors of the light were interpreted appropriately and were mostly not concerned about being observed.

8.5.5 Automatic State Changes and Accuracy

The algorithm of the FlowLight caused automatic state changes to indicate a user is, for instance, available for interruptions or busy and not interruptible. Figure 8.4b illustrates the change in distribution of the number of state changes per participant per day before and after installation. A Wilcoxon signed-rank test with Bonferroni-adjusted alpha levels of .01 per test (.05/5) showed a statistically significant change in the number of state changes (Z =-5.5337, $p \leq .01$) with an increase in state changes from 1.8 before to 8.4 after. This increase shows that the automated algorithm is affecting users' availability status in Skype.

Figure 8.6: Time Spent in each State Before (Pre) and After (Post) Installation.

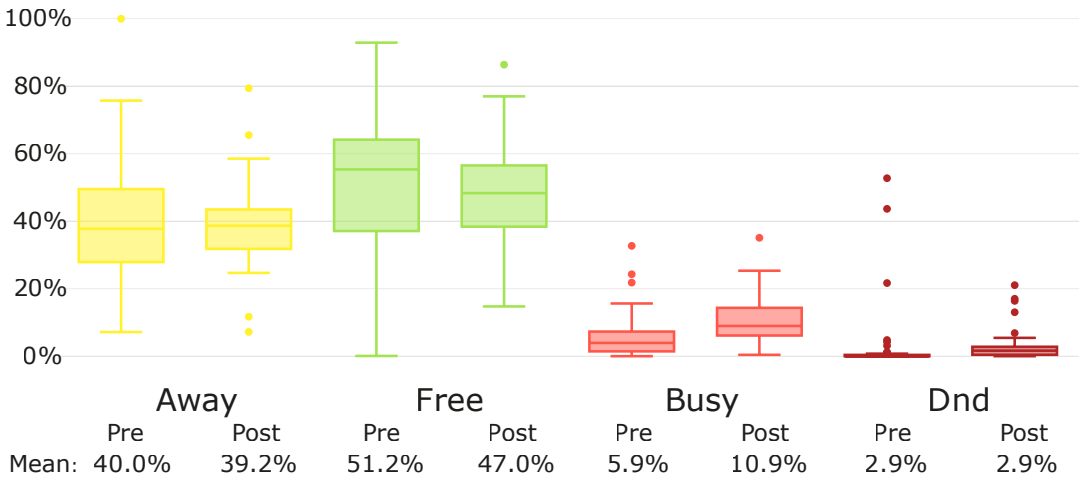


Figure 8.6 presents the time spent in each state before and after light installation. Analysis of this data shows a small insignificant decrease in time spent in the available state from 51.2% to 47.0% (Wilcoxon signed-rank test Z =-1.7143, p = .043) and a significant increase in the time spent in the busy state from 5.9% to 10.9% (Z =-3.6403, $p \leq .01$), a very small yet significant difference in do not disturb state (Z =-3.2093, $p \leq .01$), and no significant changes to time spent in the away state. Note that during the before-light period the status was already affected by meetings entered in the calendar, which caused the status to change to busy.

Participants generally agreed that the FlowLight captured their state of availability for interruptions accurately:

> "*I think it [state representativeness] was actually quite good, because what I found is, if I'm not working on a critical task, for example, responding to email which usually isn't critically mind provoking. The light would be green and then people would take that opportunity to stop by and see what they needed to talk about. Whereas if I was in the middle of a meeting or if I was more involved in my work, it would turn red and then at that point they might wait for it to turn green. That's my impression.*" - I20

Overall, 71.0% of survey respondents agreed that the FlowLight captures their state accurately while only 15.8% disagreed (Figure 8.5). This shows that even an interruptibility measure based on a simple algorithm might be accurate enough to be accepted by users and provide value.

At the same time, interview participants and 64% of our survey respondents mentioned that there are situations where the FlowLight was not representative and accurate, partly stemming from limitations in measuring interruptibility solely with computer interaction data:

> "*The light was mostly green while debugging code. During debugging, I think interrupts hurt a lot. On the other hand, the light was sometimes red when working on documents / e-mails that do not require too much focus.*" - S45

> "*[The] light captures the movements of the mouse and keyboard, and actually, there are times, which I think of a solution separate from the time, which I implement [it] so ... I'm the most occupied when I think something and usually, I write it on a paper or just keep it on my mind.*" - I4

In several cases, participants just changed to setting the state manually when it was not accurate and they wanted to indicate to others that they are available or do not want to be disturbed:

> "*There was a case when I was reading an article, and I needed a 100% concentration on that, so I just manually changed my status to busy. It was helping me a lot. I think my colleagues are also doing the same when they are engrossed in an article and they want free time, they'll just keep their light busy.*" - I4

In fact, 32% of our survey respondents reported to have changed their Skype status (which is linked to the FlowLight) more often after the light was installed, 23% less often and 45% had no changes. With the FlowLight installed, 17% of participants reported to change their status at least once a day, 37% one to several times a week, and 46% rarely or never. The job role can also affect the accuracy of the FlowLight, especially for managers, administrative assistants, and sales people. For instance, several managers mentioned that interaction was such a core part of their role that they felt they should always be available and turned off the automatic feature.

8.5.6 Continued Usage of FlowLight

Most participants, 82.6% of the 23 interview participants and 79.1% of the 183 survey participants, stated their intention to keep using the FlowLight even after the pilot period. This sentiment is reflected in actual usage data: two months after installing the FlowLight application 85.5% of users remained active (384/449).

Based on online logging of application instances that we started in November 2015, Figure 8.2 shows the number of active FlowLight users per day. The Figure also depicts the start date and relative size for the major pilots (*e.g.*, India #1 started in August '15 and had 80 participants, Norway started in November '14 and had 44).

Note that due to holidays in different locales, vacation, sick days, and travel the number of active users per day is consistently about 70% of the number of unique users over the last month (*e.g.*, a measure of 200 active users per day indicates about 315 number of unique users in the last month).

In spite of most users continuing to use the FlowLight, about 20% of users discontinued usage. There were several reasons that we identified from the interviews and surveys that decreased the benefit of the FlowLight, including the office layout and the visibility of the LED light, the company culture and people ignoring the lights, the initial willingness to use such a system, and the accuracy of the state indicated by the FlowLight. In some cases, the decreased benefit also resulted in participants ceasing to use the FlowLight:

> "*From my perspective that was something I was against from the first day but as I said I decided to join the pilot because I am a team member. ... From time to time I was looking at it but it was a little bit discouraging because the color of the light didn't reflect what I was doing and maybe after one week of using it I gave up totally.*" - I9

8.5.7 Professional Differences in Using the FlowLight

An analysis of the survey responses with respect to professional roles shows that developers (including testers) and project managers stated more frequently than

participants from other working areas that they wanted to continue using the FlowLight , even though not significantly (82% vs 70% on average) and perceived their state to be significantly more accurate (77% vs 60%, t =2.51, p =.01). For project managers, these differences might be explained by the fact that they also reported more often (but not significantly) to manually change their FlowLight status on a daily basis than participants from other work areas (24% vs 16%) and by our experiences gathered during the installation phase, in which managers often asked to disable the automatic mode completely as they wanted to be available for most of their work time. For developers, the differences might be explained by their extensive computer interaction, but future research is needed to confirm this.

8.6 Discussion

The results of our large-scale and long-term study show that the FlowLight can reduce the interruption costs for knowledge workers and can increase the awareness, amongst other benefits. In the following, we discuss implications of our findings, in particular with respect to the combination of the physical indicator with the automatic interruptibility measure, the accuracy of the measure, and the cost of not interrupting. Finally, we discuss threats to validity and limitations of our study.

8.6.1 Reasons for FlowLight's Positive Effects

The FlowLight uses a combination of a physical LED light with an automatic measure based on computer interaction to update the user's interruptibility status. The findings show that the approach was well adopted and successfully reduced in-person interruption costs. This poses the question if these effects might after all stem solely either from the automatic interruptibility measure or the physical LED light. With respect to the sole use of an automatic interruptibility measure, prior related work that used an automatic measure to update computer-based contact-list style tools, did not find any or the same level of positive effects

as our study on both, cost reduction and awareness [Begole et al., 2004; Lai et al., 2003; Tang et al., 2001]. On the other hand, manually maintaining the interruptibility state incurs a high cost as shown by previous research [Milewski and Smith, 2000] and only very few of our users switched to the manual option in cases the algorithm was not accurate enough or they wanted to ensure some undisrupted time. In addition, our findings show that while participants have a high tolerance for the accuracy of the automatic interruptibility status updates, when inaccuracies happen too often, participants also stop using the approach altogether. Overall, this indicates that the combination of the physical LED light and the automatic interruptibility measure is important to provide significant benefits to knowledge workers to use it in the long-term and that it led to the positive impact on awareness and interruption cost found in our study.

8.6.2 Accuracy of Automatic Interruptibility Measure

Participants' high tolerance for the accuracy of the automatic interruptibility measure of the FlowLight poses the question of how accurate the underlying measure has to be to provide sufficient benefit to the user. Over the course of our field study, we adapted the automatic measure two times to account for early user feedback, yet we did not find any significant differences in the effects on interruption cost and behavior. However, we intend to study the relation between accuracy and the effects on interruption cost further in the future.

Also, while participants had a high tolerance, they reported numerous situations in which they observed the status to be set incorrectly. The most frequent situation in which the status is incorrect occurs when participants "think" about something and experience a high cognitive load, yet do not interact with the computer at all. In future work and with the continuously decreasing invasiveness of biometric sensors, we plan to extend our approach to integrate biometric sensors, to cover these situations more accurately. We further plan to improve our algorithm by integrating application data, which we were not able to collect in this study due to privacy constraints. Knowing the current application might improve the algorithm's accuracy, *e.g.*, one might be less interruptible while working in a development related program and more while being in an email

client. As the nature of work and interactions vary across work areas and job roles, tailoring the algorithm accordingly could further improve its accuracy.

8.6.3 Cost of Not Interrupting

As related work has shown, not all interruptions are bad and some are definitely needed, for instance, to unblock co-workers. By physically indicating knowledge workers as not interruptible (Busy and DnD state), the FlowLight might prevent co-workers from interrupting them for important issues, reducing overall team productivity. The findings of our study on the FlowLight provides evidence that this cost is minimal at best for two reasons. First, a data analysis of the usage logs collected for our study shows that the FlowLight ends up having a significant yet small effect on the time that a knowledge worker is indicated as not interruptible (+5% per day). Second, while the FlowLight increases the awareness of the cost of interruptions, participants still interrupt their co-workers regardless of the FlowLight state if they have an important concern to discuss, as also stated by 35% of our interview participants, without being explicitly asked.

8.6.4 Threats and Limitations

A major threat to the validity of our study is the completeness of the collected data. For instance, we were not able to identify participants across different data sets. While we encouraged participants to share their data and ensured them that we only use it for research purposes, we could not demand it due to privacy concerns. We were also not able to collect geographic data due to privacy concerns and thus were not able to analyze geographic differences.

Similarly, the accuracy of the interruption logs might be incomplete or not completely accurate. Since interruption logs are based on self-reports, participants might have forgotten to log some interruptions. Also, the work patterns and habits of the days on which they logged interruptions before and after the installation of the FlowLight might have been significantly different, which makes it more difficult to compare the effect of the FlowLight. We tried to mitigate this risk by only including the logs of participants who logged interruptions for more

than three days before and three days after and by regularly reminding them to log their interruptions. Furthermore, different participants might have different criteria and judgement standards for logging interruptions. We tried to mitigate this fact by instructing participants to only log external in-person interruptions at work. In addition, by using a paired test that only compares within subject (Wilcoxon signed rank), we mitigate this effect as long as participants did not change their definition of an interruption over time.

We limited the validity threats related to generalizability across individuals and teams by collecting data from 449 participants from twelve countries and with a variety of job roles. As not all participants are native English speakers, there might be a response bias. We tried to mitigate this risk by providing sufficient instructions, opportunity for contacting us if participants had any questions, and also by visiting each major pilot site to introduce and explain the study. Based on the large number and diversity of participants, we observed that responses were not dominantly distributed to extremes, which would indicate that these knowledge workers were particularly biased based on such difficulties. From our in-person experience we can report that with very few exceptions we perceived similar acceptance, respect and in general a very positive perception of the FlowLight across all locations.

Another threat is the influence of the various algorithms on the study results. Since we wanted to ensure that participants are satisfied with the FlowLight and that we take their feedback serious, we evolved the algorithm two times. To mitigate the risk of a certain bias in the data, we looked for significant differences between populations where we might expect to find them and did not find any.

8.7 Conclusion

In-person interruptions at the workplace can incur a high cost and consume a lot of a knowledge worker's time, if they happen at inopportune moments. While there are several approaches to possibly reduce the interruption costs, little is known about the impact of a physical and automatic interruptibility indicator. In this paper, we presented FlowLight—an automatic interruptibility indicator

in the form of a physical traffic-light like LED—and reported on results from a large-scale and long-term field study with 449 participants from 12 countries. We found that the FlowLight significantly reduced the number of interruptions by 46%. We also observed an increased awareness of the potential disruptiveness of interruptions at inopportune moments, which impacts the interaction culture in a positive way, and that our approach can motivate knowledge workers and make them feel more productive. We discuss the importance of combining the physical indicator with the automatic interruptibility measure and the high tolerance of participants to the accuracy of the approach. Overall, our study provides deep insights and strong evidence on the very positive effects of the long-term usage of the FlowLight, and the continued usage of the approach by most participants indicates the success of the approach.

8.8 Acknowledgments

The authors would like to thank all study participants. This work was funded in part by SNF.

Bibliography

Agapie, E., Avrahami, D., and Marlow, J. (2016). Staying the Course: System-Driven Lapse Management for Supporting Behavior Change. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*.

Agarwal, R. and Karahanna, E. (2000). Time Flies When You're Having Fun: Cognitive Absorption and Beliefs about Information Technology Usage. *MIS Quarterly*, 24(4):665–694.

Albrecht, A. J. (1979). Measuring application development productivity. In *IBM Conference on Application Development*, pages 83–92.

Allen, D. (2015). *Getting things done: The art of stress-free productivity*. Penguin.

Althoff, T., Horvitz, E., White, R. W., and Zeitzer, J. (2017). Harnessing the web for population-scale physiological sensing: A case study of sleep and performance. In *Proceedings of the 26th International Conference on World Wide Web*, pages 113–122. International World Wide Web Conferences Steering Committee.

Altmann, E. M. and Trafton, J. G. (2002). Memory for goals: An activation-based model. *Cognitive science*, 26(1):39–83.

Altmann, E. M. and Trafton, J. G. (2004). Task interruption: Resumption lag and the role of cues.

Amabile, T. and Kramer, S. (2011). *The progress principle: Using small wins to ignite joy, engagement, and creativity at work*. Harvard Business Press.

Amann, S., Proksch, S., Nadi, S., and Mezini, M. (2016). A study of visual studio usage in practice. In *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER '16)*.

Ancker, J. S. and Kaufman, D. (2007). Rethinking health numeracy: A multidisciplinary literature review. *Journal of the American Medical Informatics Association*, 14(6):713–721.

Andreessen, M. (2011). Why software is eating the world. The Wall Street Journal, August 20, 2011.

Anvik, J., Hiew, L., and Murphy, G. C. (2006). Who should fix this bug? In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 361–370. ACM.

Arciniegas-Mendez, M., Zagalsky, A., Storey, M.-A., and Hadwin, A. F. (2017). Using the model of regulation to understand software development collaboration practices and tool support. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, pages 1049–1065. ACM.

Arroyo, E. and Selker, T. (2011). Attention and intention goals can mediate disruption in human-computer interaction. In *IFIP Conference on Human-Computer Interaction*, pages 454–470. Springer.

Astromskis, S., Bavota, G., Janes, A., Russo, B., and Penta, M. D. (2017). Patterns of developers' behaviour: A 1000-hour industrial study. *Journal of Systems and Software*, 132:85–97.

Babar, M. A., Kitchenham, B., and Jeffery, R. (2008). Comparing distributed and face-to-face meetings for software architecture evaluation: A controlled experiment. *Empirical Software Engineering*, 13(1):39–62.

Babbie, E. (2015). *The practice of social research*. Nelson Education.

Bacchelli, A. and Bird, C. (2013). Expectations, Outcomes, and Challenges of Modern Code Review. In *Proceedings of the 2013 International Conference on Software Engineering*, pages 712–721.

Bailey, B. P. and Iqbal, S. T. (2008). Understanding changes in mental workload during execution of goal-directed tasks and its application for interruption management. *ACM Transactions on Computer-Human Interaction (TOCHI)*, 14(4):21.

Bailey, B. P. and Konstan, J. A. (2006). On the need for attention-aware systems: Measuring effects of interruption on task performance, error rate, and affective state. *Computers in human behavior*, 22(4):685–708.

Bailey, B. P., Konstan, J. A., and Carlis, J. V. (2001). The effects of interruptions on task performance, annoyance, and anxiety in the user interface. In *Interact*, volume 1, pages 593–601.

Balog, K., Azzopardi, L., and de Rijke, M. (2006). Formal models for expert finding in enterprise corpora. In *Proceedings of the 29th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '06, pages 43–50. ACM.

Baltes, S. and Diehl, S. (2018). Towards a theory of software development expertise. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pages 187–200. ACM.

Bao, L., Xing, Z., Xia, X., Lo, D., and Hassan, A. E. (2018). Inference of development activities from interaction with uninstrumented applications. *Empirical Software Engineering*, 23(3):1313–1351.

Barker, C. (2003). *Cultural studies: Theory and practice*. Sage.

Barley, S. R., Meyerson, D. E., and Grodal, S. (2011). E-mail as a source and symbol of stress. *Organization Science*, 22(4):887–906.

Bartram, L. (2015). Design challenges and opportunities for eco-feedback in the home. *IEEE Computer Graphics and Applications*, 35(4).

Basseville, M., Nikiforov, I. V., et al. (1993). *Detection of abrupt changes: theory and application*, volume 104. Prentice Hall Englewood Cliffs.

Baumer, E. P., Khovanskaya, V., Matthews, M., Reynolds, L., Schwanda Sosik, V., and Gay, G. (2014). Reviewing reflection: On the use of reflection in interactive system design. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 93–102. ACM.

Beecham, S., Baddoo, N., Hall, T., Robinson, H., and Sharp, H. (2008). Motivation in software engineering: A systematic literature review. *Information and Software Technology*, 50(9):860 – 878.

Begole, J. B., Matsakis, N. E., and Tang, J. C. (2004). Lilsys: sensing unavailability. In *Proceedings of the 2004 ACM conference on Computer supported cooperative work*, pages 511–514. ACM.

Begole, J. B., Tang, J. C., Smith, R. B., and Yankelovich, N. (2002). Work Rhythms: Analyzing Visualizations of Awareness Histories of Distributed Groups. 230.

Beller, M., Gousios, G., Panichella, A., Proksch, S., Amann, S., and Zaidman, A. (2017). Developer Testing in The IDE: Patterns, Beliefs, and Behavior. *IEEE Transactions on Software Engineering*, 14(8):1–23.

Beller, M., Levaja, I., Panichella, A., Gousios, G., and Zaidman, A. (2016). How to catch 'em all: Watchdog, a family of ide plug-ins to assess testing. In *2016 IEEE/ACM 3rd International Workshop on Software Engineering Research and Industrial Practice (SER IP)*, pages 53–56.

Bellezza, F. S. and Hartwell, T. C. (1981). Cuing subjective units. *The Journal of Psychology*, 107(2):209–218.

Benjamini, Y. and Hochberg, Y. (1995). Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the royal statistical society. Series B (Methodological)*, pages 289–300.

Bentley, F., Tollmar, K., Stephenson, P., and Levy Laura (2013). Health Mashups: Presenting Statistical Patterns between Wellbeing Data and Context in Natural Language to Promote Behavior Change. 20(5):1–27.

Biehl, J. T., Czerwinski, M., Smith, G., and Robertson, G. G. (2007). Fastdash: A visual dashboard for fostering awareness in software teams. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 1313–1322. ACM.

Bjelica, M. Z., Mrazovac, B., Papp, I., and Teslic, N. (2011). Busy flag just got better: Application of lighting effects in mediating social interruptions. In *MIPRO, 2011 Proceedings of the 34th International Convention*, pages 975–980. IEEE.

Blackburn, J., Scudder, G., and Van Wassenhove, L. (1996). Improving speed and productivity of software development: a global survey of software developers. *IEEE Transactions on Software Engineering*, 22(12):875–885.

Blair, E. and Burton, S. (1987). Cognitive processes used by survey respondents to answer behavioral frequency questions. *Journal of consumer research*, 14(2):280–288.

Blynclight, E. (2019). `http://www.embrava.com/`. retrieved March 12, 2019.

Boehm, B. W. (1987). Improving software productivity. In *Computer*. Citeseer.

Boehm, B. W., Madachy, R., Steece, B., et al. (2000). *Software cost estimation with Cocomo II with Cdrom*. Prentice Hall PTR.

Borst, J. P., Taatgen, N. A., and van Rijn, H. (2015). What makes interruptions disruptive?: A process-model account of the effects of the problem state bottleneck on task interruption and resumption. In *Proceedings of the 33rd*

annual ACM conference on human factors in computing systems, pages 2971–2980. ACM.

Bradburn, N. M. (2010). Recall period in consumer expenditure surveys program.

Bradburn, N. M., Rips, L. J., and Shevell, S. K. (1987). Answering autobiographical questions: The impact of memory and inference on surveys. *Science*, 236(4798):157–161.

Bradley, N. C., Fritz, T., and Holmes, R. (2018). Context-aware conversational developer assistants. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018*, pages 993–1003.

Bragdon, A., Reiss, S. P., Zeleznik, R., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., and LaViola, Jr., J. J. (2010a). Code bubbles: Rethinking the user interface paradigm of integrated development environments. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 455–464. ACM.

Bragdon, A., Zeleznik, R., Reiss, S. P., Karumuri, S., Cheung, W., Kaplan, J., Coleman, C., Adeputra, F., and LaViola, Jr., J. J. (2010b). Code bubbles: A working set-based interface for code understanding and maintenance. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2503–2512. ACM.

Brath, R. and Peters, M. (2004). Dashboard design: Why design is important. *DM Direct*, 85.

Braun, V. and Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative research in psychology*, 3:77–101.

Bravata, D. M., Smith-Spangler, C., Sundaram, V., Gienger, A. L., Lin, N., Lewis, R., Stave, C. D., Olkin, I., and Sirard, J. R. (2007). Using pedometers to increase physical activity and improve health: A systematic review. *Jama*, 298(19):2296–2304.

Brdiczka, O., Su, N. M., and Begole, J. B. (2010). Temporal task footprinting: identifying routine tasks by their temporal patterns. In *Proceedings of the 15th international conference on Intelligent user interfaces*, pages 281–284. ACM.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Brockbank, A. and McGill, I. (2007). *Facilitating reflective learning in higher education*. McGraw-Hill Education (UK).

Brooks Jr, F. P. (1995). *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition, 2/E*. Pearson Education India.

Brown, A. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *Journal of the Learning Sciences*, 2(2):141–178.

Brown, C., Efstratiou, C., Leontiadis, I., Quercia, D., and Mascolo, C. (2014). Tracking serendipitous interactions: How individual cultures shape the office. pages 1072–1081.

Bruegge, B. and Dutoit, A. H. (2004). *Object-Oriented Software Engineering Using UML, Patterns and Java-(Required)*, volume 2004. Prentice Hall.

Burnett, M., Stumpf, S., Macbeth, J., Makri, S., Beckwith, L., Kwan, I., Peters, A., and Jernigan, W. (2016). Gendermag: A method for evaluating software's gender inclusiveness. *Interacting with Computers*, 28(6):760–787.

Calvo, R. A. and Peters, D. (2014). *Self-Awareness and Self-Compassion*, page 304. MIT Press.

Card, S. K. and Henderson Jr, A. (1986). A multiple, virtual-workspace interface to support user task switching. *ACM SIGCHI Bulletin*, 17(SI):53–59.

Cataldo, M., Herbsleb, J. D., and Carley, K. M. (2008). Socio-technical congruence: A framework for assessing the impact of technical and work dependencies on software development productivity. In *Proceedings of the Second ACM-IEEE*

International Symposium on Empirical Software Engineering and Measurement, ESEM '08, pages 2–11. ACM.

Chalmers, T. C., Smith Jr, H., Blackburn, B., Silverman, B., Schroeder, B., Reitman, D., and Ambroz, A. (1981). A method for assessing the quality of a randomized control trial. *Controlled clinical trials*, 2(1):31–49.

Charmaz, K. (2014). *Constructing grounded theory*. Sage.

Chatzoglou, P. D. and Macaulay, L. A. (1997). The importance of human factors in planning the requirements capture stage of a project. *International Journal of Project Management*, 15(1):39–53.

Chen, D., Hart, J., and Vertegaal, R. (2007). Towards a physiological model of user interruptability. In *IFIP Conference on Human-Computer Interaction*, pages 439–451. Springer.

Choe, E. K., Lee, N. B., Lee, B., Pratt, W., and Kientz, J. a. (2014). Understanding quantified-selfers' practices in collecting and exploring personal data. *Proceedings of the 32nd annual ACM conference on Human factors in computing systems (CHI '14)*, pages 1143–1152.

Chong, J. and Siino, R. (2006). Interruptions on software teams: a comparison of paired and solo programmers. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 29–38. ACM.

Claes, M., Mäntylä, M. V., Kuutila, M., and Adams, B. (2018). Do programmers work at night or during the weekend? In *Proceedings of the 40th International Conference on Software Engineering*, pages 705–715. ACM.

Claessens, B., Eerde, W., G. Rutte, C., and Roe, R. (2010). Things to do today...: A daily diary study on task completion at work. *Applied Psychology*, 59:273 – 295.

Clear, J. (2018). *Atomic Habits: Tiny Changes, Remarkable Results*. Penguin Publishing Group.

Codealike (2019). `http://codealike.com`. Retrieved March 19, 2019.

Collins, E. I. M., Cox, A. L., Bird, J., and Harrison, D. (2014). Social networking use and rescuetime: The issue of engagement. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, UbiComp '14 Adjunct, pages 687–690. ACM.

Coman, I. D. and Sillitti, A. (2008). Automated identification of tasks in development sessions. In *2008 16th IEEE International Conference on Program Comprehension*, pages 212–217.

Consolvo, S., Klasnja, P., McDonald, D. W., Avrahami, D., Froehlich, J., LeGrand, L., Libby, R., Mosher, K., and Landay, J. A. (2008a). Flowers or a robot army?: Encouraging awareness & activity with personal, mobile displays. In *Proceedings of the 10th International Conference on Ubiquitous Computing*, UbiComp '08, pages 54–63. ACM.

Consolvo, S., McDonald, D. W., and Landay, J. A. (2009). Theory-driven design strategies for technologies that support behavior change in everyday life. In *Proceedings of the SIGCHI conference on human factors in computing systems*, pages 405–414. ACM.

Consolvo, S., McDonald, D. W., Toscos, T., Chen, M. Y., Froehlich, J., Harrison, B., Klasnja, P., LaMarca, A., LeGrand, L., Libby, R., Smith, I., and Landay, J. A. (2008b). Activity sensing in the wild: a field trial of ubifit garden. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1797–1806. ACM.

Consolvo, S. and Walker, M. (2003). Using the experience sampling method to evaluate ubicomp applications. *IEEE Pervasive Computing*, 2(2):24–31.

Cordeiro, F., Bales, E., Cherry, E., and Fogarty, J. (2015). Rethinking the mobile food journal: Exploring opportunities for lightweight photo-based capture. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 3207–3216. ACM.

Couger, J. D., Adelsberger, H., Borovits, I., Zviran, M., and Motiwalla, J. (1990). Commonalities in motivating environments for programmer/analysts in austria, israel, singapore, and the usa. *Information & Management*, 18(1):41–46.

Csikszentmihalyi, M. and Larson, R. (2014). Validity and reliability of the experience-sampling method. In *Flow and the foundations of positive psychology*, pages 35–54. Springer.

Cutrell, E., Czerwinksi, M., and Horvitz, E. (2000). Notification, Disruption, and Memory: Effects of Messaging Interruptions on Memory and Performance. (1999).

Czerwinski, M., Cutrell, E., and Horvitz, E. (2000). Instant messaging: Effects of relevance and timing. In *People and Computers XIV: Proceedings of HCI 2000*, volume 2, pages 71–76.

Czerwinski, M., Horvitz, E., and Wilhite, S. (2004). A diary study of task switching and interruptions. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 175–182. ACM.

Dabbish, L. A. and Kraut, R. E. (2006). Email overload at work: an analysis of factors associated with email strain. In *Proceedings of the 2006 20th anniversary conference on Computer supported cooperative work*, pages 431–440. ACM.

Danaher, K. and Crandall, C. S. (2008). Stereotype threat in applied settings re-examined. *Journal of Applied Social Psychology*, 38(6):1639–1655.

Daniel, W. W. and Cross, C. L. (2018). *Biostatistics: a foundation for analysis in the health sciences*. Wiley.

Daskalova, N., Metaxa-Kakavouli, D., Tran, A., Nugent, N., Boergers, J., McGeary, J., and Huang, J. (2016). SleepCoacher: A Personalized Automated Self-Experimentation System for Sleep Recommendations. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 347–358.

DeMarco, T. and Lister, T. (1985). Programmer performance and the effects of the workplace. In *Proceedings of the 8th international conference on Software engineering*, pages 268–272. IEEE Computer Society Press.

DeMarco, T. and Lister, T. (2013). *Peopleware: productive projects and teams*. Addison-Wesley.

DeskTime (2019). `https://desktime.com`. Retrieved January 28, 2019.

Devanbu, P., Karstu, S., Melo, W., and Thomas, W. (1996). Analytical and empirical evaluation of software reuse metrics. In *Proceedings of the 18th International Conference on Software Engineering*, ICSE '96, pages 189–199. IEEE.

Di Stefano, G., Gino, F., Pisano, G. P., and Staats, B. R. (2016). Making experience count: The role of reflection in individual learning. *Harvard Business School NOM Unit Working Paper*, (14-093):14–093.

Dodd, N. G. and Ganster, D. C. (1996). The interactive effects of variety, autonomy, and feedback on attitudes and performance. *Journal of Organizational Behavior*, 17(4):329–347.

Donohue, B. (2018). Three-day no-meeting schedule for engineers. `https://medium.com/@Pinterest_Engineering/three-day-no-meeting-schedule-for-engineers-fca9f857a567`.

Doran, G. T. (1981). There's a smart way to write management's goals and objectives. *Management review*, 70(11):35–36.

Dragunov, A. N., Dietterich, T. G., Johnsrude, K., McLaughlin, M., Li, L., and Herlocker, J. L. (2005). Tasktracer: A desktop environment to support multi-tasking knowledge workers. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, pages 75–82. ACM.

Edelson, M., Sharot, T., Dolan, R. J., and Dudai, Y. (2011). Following the crowd: brain substrates of long-term memory conformity. *science*, 333(6038):108–111.

Enns, H. G., Ferratt, T. W., and Prasad, J. (2006). Beyond stereotypes of it professionals: Implications for it hr practices. *Commun. ACM*, 49(4).

Epstein, D. A., Avrahami, D., and Biehl, J. T. (2016a). Taking 5: Work-Breaks, Productivity, and Opportunities for Personal Informatics for Knowledge Workers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*.

Epstein, D. A., Caraway, M., Johnston, C., Ping, A., Fogarty, J., and Munson, S. A. (2016b). Beyond Abandonment to Next Steps: Understanding and Designing for Life After Personal Informatics Tool Use. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 1109–1113.

Epstein, D. A., Ping, A., Fogarty, J., and Munson, S. A. (2015). A lived informatics model of personal informatics. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 731–742. ACM.

Fisher, C. D. (2000). Mood and emotions while working: missing pieces of job satisfaction? *Journal of Organizational Behavior*, 21(2):185–202.

Fitbit (2019). `http://fitbit.com`. Retrieved March 19, 2019.

Flyvbjerg, B. (2006). Five misunderstandings about case-study research. *Qualitative inquiry*, 12(2):219–245.

Fogarty, J., Ko, A. J., Aung, H. H., Golden, E., Tang, K. P., and Hudson, S. E. (2005). Examining task engagement in sensor-based statistical models of human interruptibility. *Proceedings of the SIGCHI conference on Human factors in computing systems - CHI '05*, page 331.

Fogarty, J., Lai, J., and Christensen, J. (2004). Presence versus availability: the design and evaluation of a context-aware communication client. *International Journal of Human-Computer Studies*, 61(3):299–317.

Fogg, B. J. (2003). *Persuasive Technology: Using Computers to Change What We Think and Do*. Interactive Technologies. Elsevier Science.

Fredrickson, B. L. (1998). What good are positive emotions? *Review of general psychology*, 2(3):300.

Fredrickson, B. L., Cohn, M. A., Coffey, K. A., Pek, J., and Finkel, S. M. (2008). Open hearts build lives: positive emotions, induced through loving-kindness meditation, build consequential personal resources. *Journal of personality and social psychology*, 95(5):1045.

Fritz, T., Huang, E. M., Murphy, G. C., and Zimmermann, T. (2014). Persuasive technology in the real world: A study of long-term use of activity sensing devices for fitness. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 487–496. ACM.

Fritz, T. and Murphy, G. C. (2010). Using information fragments to answer the questions developers ask. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*, volume 1, page 175. ACM Press.

Galesic, M. and Garcia-Retamero, R. (2011). Graph literacy a cross-cultural comparison. *Medical Decision Making*, 31(3):444–457.

Gasser, R., Brodbeck, D., Degen, M., Luthiger, J., Wyss, R., and Reichlin, S. (2006). Persuasiveness of a mobile lifestyle coaching application using social facilitation. In *International Conference on Persuasive Technology*, pages 27–38. Springer.

Goler, L., Gale, J., and Grant, A. (2019). Let's not kill performance evaluations yet. `https://hbr.org/2016/11/lets-not-kill-performance-evaluations-yet`.

Gonçalves, M. K., de Souza, L., and González, V. M. (2011). Collaboration, information seeking and communication: An observational study of software

developers' work practices. *Journal of Universal Computer Science*, 17(14):1913–1930.

González, V. M. and Mark, G. (2004). Constant, constant, multi-tasking craziness: Managing multiple working spheres. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 113–120. ACM.

Graziotin, D., Fagerholm, F., Wang, X., and Abrahamsson, P. (2017). On the Unhappiness of Software Developers. pages 324–333.

Graziotin, D., Wang, X., and Abrahamsson, P. (2014a). Happy software developers solve problems better: psychological measurements in empirical software engineering. *PeerJ*, 2:e289.

Graziotin, D., Wang, X., and Abrahamsson, P. (2014b). Software developers, moods, emotions, and performance. *IEEE Software*, 31(4):24–27.

Graziotin, D., Wang, X., and Abrahamsson, P. (2015a). Do feelings matter? on the correlation of affects and the self-assessed productivity in software engineering. *Journal of Software: Evolution and Process*, 27(7):467–487.

Graziotin, D., Wang, X., and Abrahamsson, P. (2015b). How do you feel, developer? an explanatory theory of the impact of affects on programming performance. *PeerJ Computer Science*, 1:e18.

Guest, G., Bunce, A., and Johnson, L. (2006). How many interviews are enough? an experiment with data saturation and variability. *Field methods*, 18(1):59–82.

Gustafson, K. L. and Bennett Jr, W. (2002). Promoting learner reflection: Issues and difficulties emerging from a three-year study. Technical report, Georgia University Athens, Department of Instructional Technology.

Gustafsson, F. and Gustafsson, F. (2000). *Adaptive filtering and change detection*, volume 1. Citeseer.

Hassib, M., Khamis, M., Friedl, S., Schneegass, S., and Alt, F. (2017). Brainatwork: Logging cognitive engagement and tasks in the workplace using electroencephalography. In *Proceedings of the 16th International Conference on Mobile and Ubiquitous Multimedia*, MUM '17, pages 305–310. ACM.

Herrmanny, K., Ziegler, J., and Dogangün, A. (2016). Supporting users in setting effective goals in activity tracking. In *Proceedings of the 11th International Conference on Persuasive Technology - Volume 9638*, Persuasive 2016, pages 15–26. Springer-Verlag.

Hincapié-Ramos, J. D., Voida, S., and Mark, G. (2011a). A design space analysis of availability-sharing systems. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 85–96. ACM.

Hincapié-Ramos, J. D., Voida, S., and Mark, G. (2011b). Sharing availability information with interruptme. In *Proceedings of the 13th international conference on Ubiquitous computing*, pages 477–478. ACM.

Ho, J. and Intille, S. S. (2005). Using context-aware computing to reduce the perceived burden of interruptions from mobile devices. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 909–918. ACM.

Hofstede, G. (1994). *Cultures and Organizations: Software of the Mind : Intercultural Cooperation and Its Importance for Survival*. HarperCollins.

Hofstede, G. H. and Arrindell, W. A. (1998). *Masculinity and femininity: The taboo dimension of national cultures*, volume 3 of *Cross Cultural Psychology*. Sage.

Hollis, V., Konrad, A., and Whittaker, S. (2015). Change of Heart: Emotion Tracking to Promote Behavior Change. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI '15)*, pages 2643–2652.

Horvitz, E., Koch, P., Kadie, C. M., and Jacobs, A. (2002). Coordinate: Probabilistic forecasting of presence and availability. In *Proceedings of the Eighteenth conference on Uncertainty in artificial intelligence*, pages 224–233. Morgan Kaufmann Publishers Inc.

Huang, D., Tory, M., and Bartram, L. (2016). A Field Study of On-Calendar Visualizations. In *Proceedings of Graphics Interface 2016*, pages 13–20.

Hudson, R. L. and Davis, J. L. (1972). The effects of intralist cues, extralist cues, and category names on categorized recall. *Psychonomic Science*, 29(2):71–75.

Hudson, S., Fogarty, J., Atkeson, C., Avrahami, D., Forlizzi, J., Kiesler, S., Lee, J., and Yang, J. (2003). Predicting human interruptibility with sensors: a wizard of oz feasibility study. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 257–264. ACM.

Humphrey, W. S. (1996). Using a defined and measured personal software process. *IEEE*, 13(3):77–88.

Humphrey, W. S. (2000). The Personal Software Process (PSP). (November).

Iqbal, S. T. and Bailey, B. P. (2007). Understanding and developing models for detecting and differentiating breakpoints during interactive tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 697–706. ACM.

Iqbal, S. T. and Bailey, B. P. (2008). Effects of intelligent notification management on users and their tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 93–102. ACM.

Iqbal, S. T. and Horvitz, E. (2007). Disruption and recovery of computing tasks: Field study, analysis, and directions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 677–686. ACM.

Isaacs, E., Whittaker, S., Frohlich, D., and O'Conaill, B. (1997). Informal communication re-examined: New functions for video in supporting opportunistic encounters. *Video-mediated communication*, 997:459–485.

Jakobsen, M. R., Fernandez, R., Czerwinski, M., Inkpen, K., Kulyk, O., and Robertson, G. G. (2009). Wipdash: Work item and people dashboard for software development teams. In *IFIP Conference on Human-Computer Interaction*, pages 791–804. Springer.

John, O. P. and Srivastava, S. (1999). The big five trait taxonomy: History, measurement, and theoretical perspectives. *Handbook of personality: Theory and research*, 2(1999):102–138.

Johnson, P. M. and Disney, A. M. (1998). The personal software process: A cautionary case study. *IEEE Software*, 15(6):85–88.

Johnson, P. M., Kou, H., Agustin, J., Chan, C., Moore, C., Miglani, J., Zhen, S., and Doane, W. E. J. (2003). Beyond the personal software process: Metrics collection and analysis for the differently disciplined. In *Proceedings of the 25th International Conference on Software Engineering*, ICSE '03, pages 641–646. IEEE.

Johnson, S. M. and White, G. (1971). Self-observation as an agent of behavioral change. *Behavior Therapy*, 2(4):488–497.

Jones, C. (1994). Software metrics: good, bad and missing. *Computer*, 27(9):98–100.

Jones, S. L. and Kelly, R. (2017). Dealing With Information Overload in Multifaceted Personal Informatics Systems. *Human Computer Interaction*, pages 1–48.

Joo, B. K. B. and Park, S. (2010). Career satisfaction, organizational commitment, and turnover intention: The effects of goal orientation, organizational learning culture and developmental feedback. *Leadership & Organization Development Journal*, 31(6):482–500.

Kahn, B. E. and Isen, A. M. (1993). The influence of positive affect on variety seeking among safe, enjoyable products. *Journal of Consumer Research*, 20(2):257–270.

Kalliamvakou, E., Bird, C., Zimmermann, T., Begel, A., DeLine, R., and German, D. M. (2019). What makes a great manager of software engineers? *IEEE Transactions on Software Engineering*, 45(1):87–106.

Kandolin, I., Härmä, M., and Toivanen, M. (2001). Flexible working hours and well-being in finnland. *Journal of Human Ergology*, 30(1-2):35–40.

Kaufman, L. and Rousseeuw, P. (1987). *Clustering by means of medoids*. North-Holland.

Kay, M., Choe, E. K., Shepherd, J., Greenstein, B., Watson, N., Consolvo, S., and Kientz, J. A. (2012). Lullaby: A capture and access system for understanding the sleep environment. In *Proceedings of the 2012 ACM Conference on Ubiquitous Computing*, UbiComp '12, pages 226–234. ACM.

Kelly, A. (2008). *Changing Software Development: Learning to Become Agile*. Wiley.

Kersten, M. and Murphy, G. C. (2006). Using task context to improve programmer productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, SIGSOFT '06/FSE-14, pages 1–11. ACM.

Kersten-van Dijk, E. T., Westerink, J. H., Beute, F., and IJsselsteijn, W. A. (2017). Personal informatics, self-insight, and behavior change: A critical review of current literature. *Human–Computer Interaction*, 32(5-6):268–296.

Kevic, K. and Fritz, T. (2017). Towards activity-aware tool support for change tasks. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 171–182.

Khan, I. A., Brinkman, W.-P., and Hierons, R. M. (2011). Do moods affect programmers' debug performance? *Cognition, Technology & Work*, 13(4):245–258.

Kim, Y.-H. and Choe, E. K. (2019). Understanding Personal Productivity: How Knowledge Workers Define, Evaluate, and Reflect on Their Productivity. In

Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19, number May.

Kim, Y.-H., Jeon, J. H., Choe, E. K., Lee, B., Kim, K., and Seo, J. (2016). TimeAware: Leveraging Framing Effects to Enhance Personal Productivity. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*, pages 272–283.

Kitchenham, B. A. and Pfleeger, S. L. (2008). Personal opinion surveys. In *Guide to advanced empirical software engineering*, pages 63–92. Springer.

Klasnja, P., Consolvo, S., McDonald, D. W., Landay, J. A., and Pratt, W. (2009). Using mobile & personal sensing technologies to support health behavior change in everyday life: lessons learned. In *AMIA Annual Symposium Proceedings*, volume 2009, page 338. American Medical Informatics Association.

Klasnja, P., Consolvo, S., and Pratt, W. (2011). How to Evaluate Technologies for Health Behavior Change in HCI Research. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3063–3072.

Ko, A. J., DeLine, R., and Venolia, G. (2007). Information needs in collocated software development teams. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 344–353. IEEE Computer Society.

Kocielnik, R., Avrahami, D., Marlow, J., Lu, D., and Hsieh, G. (2018). Designing for workplace reflection: a chat and voice-based conversational agent. In *Proceedings of the 2018 on Designing Interactive Systems Conference 2018*, pages 881–894. ACM.

Koldijk, S., Staalduinen, M. V., Raaijmakers, S., and Kraaij, W. (2011). Activity-Logging for Self-Coaching of Knowledge Workers. pages 0–3.

Koldijk, S., van Staalduinen, M., Neerincx, M., and Kraaij, W. (2012). Real-time task recognition based on knowledge workers' computer activities. In

Proceedings of the 30th European Conference on Cognitive Ergonomics, ECCE '12, pages 152–159. ACM.

Krogstie, B., Prilla, M., Wessel, D., Knipfer, K., and Pammer, V. (2012). Computer support for reflective learning in the workplace: A model. pages 151–153.

Lai, J., Yoshihama, S., Bridgman, T., Podlaseck, M., Chou, P. B., and Wong, D. C. (2003). Myteam: Availability awareness through the use of sensor data. In *INTERACT*.

Lange, P. G. (2008). Interruptions and intertasking in distributed knowledge work. *NAPA Bulletin*, 30(1):128–147.

Larson, R. and Csikszentmihalyi, M. (2014). The experience sampling method. In *Flow and the foundations of positive psychology*, pages 21–34. Springer.

LaToza, T. D., Venolia, G., and DeLine, R. (2006). Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*, pages 492–501. ACM.

LeBoeuf, R. A. and Shafir, E. (2009). Anchoring on the" here" and" now" in time and distance judgments. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 35(1):81.

Lee, J., Walker, E., Burleson, W., and Hekler, E. B. (2014). Exploring users' creation of personalized behavioral plans. In *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*, pages 703–706. ACM.

Lee, J., Walker, E., Burleson, W., and Hekler, E. B. (2015). Understanding users' creation of behavior change plans with theory-based support. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, pages 2301–2306. ACM.

Lee, J., Walker, E., Burleson, W., Kay, M., Buman, M., and Hekler, E. B. (2017). Self-Experimentation for Behavior Change. In *Proceedings of the 2017*

CHI Conference on Human Factors in Computing Systems - CHI '17, pages 6837–6849.

Lemaître, G., Nogueira, F., and Aridas, C. K. (2017). Imbalanced-learn: A python toolbox to tackle the curse of imbalanced datasets in machine learning. *Journal of Machine Learning Research*, 18(17):1–5.

Li, I., Dey, A., and Forlizzi, J. (2010). A stage-based model of personal informatics systems. *Proceedings of the 28th international conference on Human factors in computing systems (CHI '10)*, page 557.

Li, I., Dey, A., and Forlizzi, J. (2011). Understanding my data, myself: supporting self-reflection with Ubicomp technologies. In *Proceedings of the 13th international conference on Ubiquitous computing (UbiComp '11)*, page 405.

Li, P. L., Ko, A. J., and Zhu, J. (2015). What makes a great software engineer? In *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, pages 700–710. IEEE Press.

Liaw, A., Wiener, M., et al. (2002). Classification and regression by randomforest. *R news*, 2(3):18–22.

Lin, J., Mamykina, L., Lindtner, S., Delajoux, G., and Strub, H. (2006). Fish'n'Steps: Encouraging Physical Activity with an Interactive Computer Game. In *UbiComp 2006: Ubiquitous Computing*, volume 4206 of *Lecture Notes in Computer Science*, chapter 16, pages 261–278.

Locke, E. A. and Latham, G. P. (1990). *A theory of goal setting & task performance.* Prentice-Hall, Inc.

Locke, E. A. and Latham, G. P. (2002). Building a practically useful theory of goal setting and task motivation: A 35-year odyssey. *American psychologist*, 57(9):705.

Lott, Y. and Chung, H. (2016). Gender discrepancies in the outcomes of schedule control on overtime hours and income in germany. *European Sociological Review*, 32(6):752–765.

Lu, D., Marlow, J., Kocielnik, R., and Avrahami, D. (2018). Challenges and opportunities for technology-supported activity reporting in the workplace. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 170:1–170:12. ACM.

Maalej, W., Ellmann, M., and Robbes, R. (2017). Using contexts similarity to predict relationships between tasks. *Journal of Systems and Software*, 128:267 – 284.

Mamykina, L., Mynatt, E., Davidson, P., and Greenblatt, D. (2008). Mahi: Investigation of social scaffolding for reflective thinking in diabetes management. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 477–486. ACM.

Manictime (2019). `http://manictime.com`. Retrieved March 19, 2019.

Marcus, B. and Schütz, A. (2005). Who are the people reluctant to participate in research? personality correlates of four different types of nonresponse as inferred from self- and observer ratings. *Journal of Personality*, 73(4):959–984.

Mark, G., Czerwinski, M., and Iqbal, S. T. (2018). Effects of Individual Differences in Blocking Workplace Distractions. In *CHI '18*. ACM.

Mark, G., Gonzalez, V. M., and Harris, J. (2005). No task left behind?: examining the nature of fragmented work. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–330. ACM.

Mark, G., Gudith, D., and Klocke, U. (2008a). The cost of interrupted work: more speed and stress. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 107–110. ACM.

Mark, G., Gudith, D., and Klocke, U. (2008b). The Cost of Interrupted Work : More Speed and Stress. In *CHI 2008: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 107–110.

Mark, G., Iqbal, S., and Czerwinski, M. (2017). How blocking distractions affects workplace focus and productivity. In *Proceedings of the 2017 ACM*

International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2017 ACM International Symposium on Wearable Computers on - UbiComp '17, pages 928–934.

Mark, G., Iqbal, S. T., Czerwinski, M., and Johns, P. (2014). Bored mondays and focused afternoons: The rhythm of attention and online activity in the workplace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 3025–3034. ACM.

Mark, G., Iqbal, S. T., Czerwinski, M., Johns, P., and Sano, A. (2016a). Email Duration, Batching and Self-interruption: Patterns of Email Use on Productivity and Stress. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*, volume 21, pages 98–109.

Mark, G., Iqbal, S. T., Czerwinski, M., Johns, P., and Sano, A. (2016b). Neurotics Can't Focus: An in situ Study of Online Multitasking in the Workplace. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1739–1744.

Mathan, S., Whitlow, S., Dorneich, M., Ververs, P., and Davis, G. (2007). Neurophysiological estimation of interruptibility: Demonstrating feasibility in a field context. In *In Proceedings of the 4th International Conference of the Augmented Cognition Society*.

Mathur, A., Broeck, M. V. D., Vanderhulst, G., Mashhadi, A., and Kawsar, F. (2015). Tiny Habits in the Giant Enterprise: Understanding the Dynamics of a Quantified Workplace. In *Proceedings of the Joint International Conference on Pervasive and Ubiquitous Computing and the International Symposium on Wearable Computers (Ubicomp/ISWC'15)*, pages 577–588.

Matthews, G. (2007). The impact of commitment, accountability, and written goals on goal achievement. In *87th Convention of the Western Psychological Association*.

Mazmanian, M. (2013). Avoiding the trap of constant connectivity: When

congruent frames allow for heterogeneous practices. *Academy of Management Journal*, 56(5):1225–1250.

McDuff, D., Karlson, A., and Kapoor, A. (2012). AffectAura: an Intelligent System for Emotional Memory. ACM.

McFarlane, D. (2002). Comparison of four primary methods for coordinating the interruption of people in human-computer interaction. *Human-Computer Interaction*, 17(1):63–139.

Melamed, S., Ben-Avi, I., Luz, J., and Green, M. S. (1995). Objective and subjective work monotony: Effects on job satisfaction, psychological distress, and absenteeism in blue-collar workers. *Journal of Applied Psychology*, 80(1):29.

Melo, C., Cruzes, D. S., Kon, F., and Conradi, R. (2011). Agile team perceptions of productivity factors. In *2011 Agile Conference*, pages 57–66. IEEE.

Menon, G. (1994). Judgments of behavioral frequencies: Memory search and retrieval strategies. In Schwarz, N. and Sudman, S., editors, *Autobiographical Memory and the Validity of Retrospective Reports*, pages 161–172. Springer.

Meyer, A., Barr, E. T., Bird, C., and Zimmermann, T. (2019). Today was a good day: The daily life of software developers. *IEEE Transactions on Software Engineering*, pages 1–1.

Meyer, A. N., Barton, L. E., Murphy, G. C., Zimmermann, T., and Fritz, T. (2017a). The Work Life of Developers: Activities, Switches and Perceived Productivity. *Transactions of Software Engineering*, pages 1–15.

Meyer, A. N., Fritz, T., Murphy, G. C., and Zimmermann, T. (2014). Software developers' perceptions of productivity. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, pages 19–29. ACM.

Meyer, A. N., Murphy, G. C., Zimmermann, T., and Fritz, T. (2017b). Design recommendations for self-monitoring in the workplace: Studies in software development. *Proc. ACM Hum.-Comput. Interact.*, 1(CSCW):79:1–79:24.

Meyer, A. N., Zimmermann, T., and Fritz, T. (2017c). Characterizing Software Developers by Perceptions of Productivity. In *Empirical Software Engineering and Measurement (ESEM), 2017 International Symposium on*.

MicrosoftGraphApi (2019). `https://graph.microsoft.io`. Retrieved March 19, 2019.

Milewski, A. E. and Smith, T. M. (2000). Providing presence cues to telephone users. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, pages 89–96. ACM.

Minelli, R., Mocci, A., and Lanza, M. (2015). I Know What You Did Last Summer – An Investigation of How Developers Spend Their Time. *Proceedings of ICPC 2015 (23rd IEEE International Conference on Program Comprehension)*, pages 25—-35.

Mintzberg, H. (1980). *The nature of managerial work*. Theory of management policy series. Prentice-Hall.

Mirza, H. T., Chen, L., Chen, G., Hussain, I., and He, X. (2011a). Switch detector: an activity spotting system for desktop. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 2285–2288. ACM.

Mirza, H. T., Chen, L., Hussain, I., Majid, A., and Chen, G. (2015). A study on automatic classification of users' desktop interactions. *Cybernetics and Systems*, 46(5):320–341.

Mirza, H. T., Chen, L., Majid, A., and Chen, G. (2011b). Building user task space by mining temporally proximate desktop actions. *Cybernetics and Systems*, 42(8):585–604.

Mockus, A. and Herbsleb, J. D. (2002). Expertise browser: a quantitative approach to identifying expertise. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002*, pages 503–512.

Monk, C. A., Trafton, J. G., and Boehm-Davis, D. A. (2008). The effect of interruption duration and demand on resuming suspended goals. *Journal of Experimental Psychology: Applied*, 14(4):299–313.

Monkaresi, H., Calvo, R., Pardo, A., Chow, K., Mullan, B., Lam, M., Twigg, S., and Cook, D. (2013). Intelligent diabetes lifestyle coach. In *OzCHI workshops programme*.

Moon, J. (2013). *Reflection in Learning and Professional Development: Theory and Practice*. Taylor & Francis.

Morisano, D., Hirsh, J. B., Peterson, J. B., Pihl, R. O., and Shore, B. M. (2010). Setting, elaborating, and reflecting on personal goals improves academic performance. *Journal of applied psychology*, 95(2):255.

Muller, J. (2018). *The Tyranny of Metrics*. Princeton University Press.

Müller, S. C. and Fritz, T. (2015). Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress. In *Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on*, volume 1, pages 688–699. IEEE.

Munson, S. and Consolvo, S. (2012). Exploring Goal-setting, Rewards, Self-monitoring, and Sharing to Motivate Physical Activity. *Proceedings of the 6th International Conference on Pervasive Computing Technologies for Healthcare*, pages 25–32.

Murgia, A., Tourani, P., Adams, B., and Ortu, M. (2014). Do developers feel emotions? an exploratory analysis of emotions in software artifacts. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, pages 262–271. ACM.

Murphy, G. C., Kersten, M., Robillard, M. P., and Čubranić, D. (2005). The emergent structure of development tasks. In *ECOOP 2005 - Object-Oriented Programming*, pages 33–48. Springer Berlin Heidelberg.

Murphy-Hill, E., Jaspan, C., Sadowski, C., Shepherd, D. C., Phillips, M., Winter, C., Dolan, A. K., Smith, E. K., and Jorde, M. A. (2019). What predicts software developers' productivity? *Transactions on Software Engineering*.

Nair, R., Voida, S., and Mynatt, E. D. (2005). Frequency-based detection of task switches. In *Proceedings of the 19th British HCI Group Annual Conference*, volume 2, pages 94–99.

Naur, P. and Randell, B. (1969). Software engineering: Report of a conference sponsored by the nato science committee. *Scientific Affairs Division, NATO*.

Nazar, N., Hu, Y., and Jiang, H. (2016). Summarizing software artifacts: A literature review. *Journal of Computer Science and Technology*, 31(5):883–909.

Neter, J. and Waksberg, J. (1964). A study of response errors in expenditures data from household interviews. *Journal of the American Statistical Association*, 59(305):18–55.

Nguyen, G. H., Bouzerdoum, A., and Phung, S. L. (2009). Learning Pattern Classification Tasks with Imbalanced Data Sets. *Pattern Recognition*.

Niemantsverdriet, K. and Erickson, T. (2017). Recurring meetings: An experiential account of repeating meetings in a large organization. *Proceedings of the ACM on Human-Computer Interaction*, 1:1–17.

Oliver, N., Czerwinski, M., Smith, G., and Roomp, K. (2008). Relalttab: assisting users in switching windows. In *IUI*.

Oliver, N., Smith, G., Thakkar, C., and Surendran, A. C. (2006). Swish: Semantic analysis of window titles and switching history. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*, IUI '06, pages 194–201. ACM.

Origo, F. and Pagani, L. (2008). Workplace flexibility and job satisfaction: Some evidence from europe. 29:539–566.

Ouweneel, E., Le Blanc, P. M., Schaufeli, W. B., and van Wijhe, C. I. (2012). Good morning, good day: A diary study on positive emotions, hope, and work engagement. *Human Relations*, 65(9):1129–1154.

OxfordDictionary (2019). Oxforddictionary. https://en.oxforddictionaries.com/definition/productivity. Retrieved February 27, 2019.

Pammer, V., Bratic, M., Feyertag, S., and Faltin, N. (2015). *The Value of Self-tracking and the Added Value of Coaching in the Case of Improving Time Management*, pages 467–472. Springer International Publishing.

Parnin, C. and DeLine, R. (2010a). Evaluating cues for resuming interrupted programming tasks. *Proceedings of the 28th international conference on Human factors in computing systems - CHI '10*, page 93.

Parnin, C. and DeLine, R. (2010b). Evaluating cues for resuming interrupted programming tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 93–102. ACM.

Parnin, C. and Rugaber, S. (2009). Resumption strategies for interrupted programming tasks. In *2009 IEEE 17th International Conference on Program Comprehension*, pages 80–89.

Parnin, C. and Rugaber, S. (2011). Resumption strategies for interrupted programming tasks. *Software Quality Journal*, 19(1):5–34.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011a). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E.

(2011b). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Perlow, L. A. (1999). The time famine: Toward a sociology of work time. *Administrative science quarterly*, 44(1):57–81.

Perry, D. E., Staudenmayer, N., and Votta, L. G. (1994a). People, organizations, and process improvement. *IEEE Softw.*, 11(4):36–45.

Perry, D. E., Staudenmayer, N. A., and Votta, L. G. (1994b). People, organizations, and process improvement. *IEEE Software*, 11(4):36–45.

Petrou, P., Demerouti, E., Peeters, M. C., Schaufeli, W. B., and Hetland, J. (2012). Crafting a job on a daily basis: Contextual correlates and the link to work engagement. *Journal of Organizational Behavior*, 33(8):1120–1141.

PomodoroTechnique (2019). pomodorotechnique.com. Retrieved March 19, 2019.

Prasad, A., Sorber, J., Stablein, T., Anthony, D., and Kotz, D. (2012). Understanding sharing preferences and behavior for mhealth devices. In *Proceedings of the 2012 ACM workshop on Privacy in the electronic society*, pages 117–128. ACM.

Prestwich, A., Perugini, M., and Hurling, R. (2009). Can the effects of implementation intentions on exercise be enhanced using text messages? *Psychology and Health*, 24(6):677–687.

Prochaska, J. O. and Velicer, W. F. (1997). The Transtheoretical Change Model of Health Behavior. *American Journal of Health Promotion*, 12(1):38–48.

Punter, T., Ciolkowski, M., Freimut, B., and John, I. (2003). Conducting on-line surveys in software engineering. In *International Symposium on Empirical Software Engineering, 2003. ISESE 2003. Proceedings.*, pages 80–88. IEEE.

Racine, J. (2000). Consistent cross-validatory model-selection for dependent data: hv-block cross-validation. *Journal of econometrics*, 99(1):39–61.

Rattenbury, T. and Canny, J. (2007). Caad: An automatic task support system. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 687–696. ACM.

Reinhardt, W., Schmidt, B., Sloep, P., and Drachsler, H. (2011). Knowledge worker roles and actions - results of two empirical studies. *Knowledge and Process Management*, 18(3):150–174.

RescueTime (2019). `http://rescuetime.com`. Retrieved March 19, 2019.

Robertson, G., Horvitz, E., Czerwinski, M., Baudisch, P., Hutchings, D. R., Meyers, B., Robbins, D., and Smith, G. (2004). Scalable fabric: Flexible task management. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, pages 85–89. ACM.

Robertson, G., Van Dantzich, M., Robbins, D., Czerwinski, M., Hinckley, K., Risden, K., Thiel, D., and Gorokhovsky, V. (2000). The task gallery: a 3d window manager. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 494–501. ACM.

Robillard, M. and Murphy, G. (2004). Program navigation analysis to support task-aware software development environments. pages 83–88.

Robillard, M., Walker, R., and Zimmermann, T. (2010). Recommendation systems for software engineering. *IEEE Software*, 27(4):80–86.

Rogelberg, S., Conway, J., E Sederburg, M., Spitzmuller, C., Aziz, S., and E Knight, W. (2004). Profiling active and passive nonrespondents to an organizational survey. *The Journal of applied psychology*, 88:1104–14.

Rožman, M., Treven, S., and Čančer, V. (2017). Motivation and satisfaction of employees in the workplace. *Business Systems Research Journal*, 8.

Rooksby, J., Asadzadeh, P., Rost, M., Morrison, A., and Chalmers, M. (2016). Personal Tracking of Screen Time on Digital Devices. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 284–296.

Rooksby, J., Rost, M., Morrison, A., and Chalmers, M. (2014). Personal tracking as lived informatics. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1163–1172.

Rule, A., Tabard, A., Boyd, K., and Hollan, J. (2015). Restoring the Context of Interrupted Work with Desktop Thumbnails. In *37th Annual Meeting of the Cognitive Science Society*, pages 1–6. Cognitive Science Society.

Ruostela, J., Lönnqvist, A., Palvalin, M., Vuolle, M., Patjas, M., and Raij, A.-L. (2015). 'new ways of working' as a tool for improving the performance of a knowledge-intensive company. *Knowledge Management Research & Practice*, 13(4):382–390.

Sach, R., Sharp, H., and Petre, M. (2011). What makes software engineers go that extra mile?

Sadowski, C. and Zimmermann, T. (2019). *Rethinking Productivity in Software Engineering*. Apress.

Safer, I. and Murphy, G. C. (2007). Comparing episodic and semantic interfaces for task boundary identification. In *Proceedings of the 2007 conference of the center for advanced studies on Collaborative research*, pages 229–243. IBM Corp.

Sahm, A. and Maalej, W. (2010). Switch! recommending artifacts needed next based on personal and shared context. In *Software Engineering (Workshops)*, pages 473–484.

Samara, A., Galway, L., Bond, R., and Wang, H. (2017). Tracking and evaluation of pupil dilation via facial point marker analysis. In *2017 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*, pages 2037–2043.

Sanchez, H., Robbes, R., and Gonzalez, V. M. (2015). An empirical study of work fragmentation in software evolution tasks. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015 - Proceedings*, pages 251–260.

Sappelli, M., Pasi, G., Verberne, S., De Boer, M., and Kraaij, W. (2016). Assessing e-mail intent and tasks in e-mail messages. *Information Sciences*, 358-359:1–17.

Sarma, A., Noroozi, Z., and Van Der Hoek, A. (2003). Palantír: raising awareness among configuration management workspaces. In *Software Engineering, 2003. Proceedings. 25th International Conference on*, pages 444–454. IEEE.

Schwarz, N. and Oyserman, D. (2001). Asking Questions About Behavior: Cognition, Communication, and Questionnaire Construction. *American Journal of Evaluation*, 22(2):127–160.

Sharot, T., Korn, C. W., and Dolan, R. J. (2011). How unrealistic optimism is maintained in the face of reality. *Nature neuroscience*, 14(11):1475.

Sharp, H., Baddoo, N., Beecham, S., Hall, T., and Robinson, H. (2009). Models of motivation in software engineering. *Information and software technology*, 51(1):219–233.

Sheldon, K. M., Ryan, R., and Reis, H. T. (1996). What makes for a good day? competence and autonomy in the day and in the person. *Personality and social psychology bulletin*, 22(12):1270–1279.

Shen, J., Irvine, J., Bao, X., Goodman, M., Kolibaba, S., Tran, A., Carl, F., Kirschner, B., Stumpf, S., and Dietterich, T. (2009). Detecting and correcting user activity switches: Algorithms and interfaces. pages 117–126.

Shen, J., Li, L., and Dietterich, T. G. (2007). Real-time detection of task switches of desktop users. In *IJCAI*, volume 7, pages 2868–2873.

Shen, J., Li, L., Dietterich, T. G., and Herlocker, J. L. (2006). A hybrid learning system for recognizing user tasks from desktop activities and email messages. In *Proceedings of the 11th International Conference on Intelligent User Interfaces*, IUI '06, pages 86–92. ACM.

Singer, J., Lethbridge, T., Vinson, N., and Anquetil, N. (2010). An examination of software engineering work practices. In *CASCON First Decade High Impact Papers*, CASCON '10, pages 174–188. IBM Corporation.

Slife (2019). `http://www.slifelabs.com`. Retrieved March 19, 2019.

Smith, G., Baudisch, P., Robertson, G., Czerwinski, M., Meyers, B., Robbins, D., and Andrews, D. (2003). Groupbar: The taskbar evolved. In *Proceedings of OZCHI*, volume 3, pages 1–10.

Soto, C. J. and John, O. P. (2016). The Next Big Five Inventory (BFI-2): Developing and Assessing a Hierarchical Model With 15 Facets to Enhance Bandwidth. 113(June):117–143.

Soules, C. A. N. and Ganger, G. R. (2005). Connections: Using context to enhance file search. In *Proceedings of the Twentieth ACM Symposium on Operating Systems Principles*, SOSP '05, pages 119–132. ACM.

Spencer, D. (2009). *Card sorting: Designing usable categories*. Rosenfeld Media.

Spira, J. B. and Feintuch, J. B. (2005). The cost of not paying attention: How interruptions impact knowledge worker productivity.

StackOverflow (2017). Stackoverflow developer survey 2015. `http://stackoverflow.com/research/developer-survey-2015`. retrieved January 16, 2017.

Steele, C. and Aronson, J. (1995). Stereotype threat and the intellectual test-performance of african-americans. *Journal of personality and social psychology*, 69:797–811.

Stern, H., Pammer, V., and Lindstaedt, S. N. (2011). A preliminary study on interruptibility detection based on location and calendar information. *Proc. CoSDEO*, 11.

Stinson, L. L. (1999). Measuring how people spend their time: a time-use survey design. *Monthly Lab. Rev.*, 122:12.

Stol, K.-J., Ralph, P., and Fitzgerald, B. (2016). Grounded theory in software engineering research. *Proceedings of the 38th International Conference on Software Engineering - ICSE '16*, (Aug 2015):120–131.

Storey, M.-A., Singer, L., Cleary, B., Figueira Filho, F., and Zagalsky, A. (2014). The (R)Evolution of Social Media in Software Engineering. In *FOSE 2014 Proceedings of the on Future of Software Engineering*, pages 100–116.

Storey, M.-A. and Zagalsky, A. (2016). Disrupting developer productivity one bot at a time. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 928–931. ACM.

Storey, M.-A., Zagalsky, A., Figueira Filho, F., Singer, L., and German, D. M. (2017). How social and communication channels shape and challenge a participatory culture in software development. *IEEE Transactions on Software Engineering*, 43(2):185–204.

Strauss, A. and Corbin, J. (1998). *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*.

Stricker, L. J. and Ward, W. C. (2004). Stereotype threat, inquiring about test takers' ethnicity and gender, and standardized test performance1. *Journal of Applied Social Psychology*, 34(4):665–693.

Stumpf, S., Bao, X., Dragunov, A., Dietterich, T. G., Herlocker, J., Johnsrude, K., Li, L., and Shen, J. (2005). Predicting user tasks: I know what you're doing. In *20th National conference on artificial intelligence (AAAI-05), workshop on human comprehensible machine learning*.

Sudman, S. and Bradburn, N. M. (1973). Effects of time and memory factors on response in surveys. *Journal of the American Statistical Association*, 68(344):805–815.

Sutherland, J., Viktorov, A., Blount, J., and Puntikov, N. (2007). Distributed scrum: Agile project management with outsourced development teams. In *Sys-*

tem Sciences, 2007. HICSS 2007. 40th Annual Hawaii International Conference on, pages 274a–274a. IEEE.

Sykes, E. R. (2011). Interruptions in the workplace: A case study to reduce their effects. *International Journal of Information Management*, 31(4):385–394.

Taillard, J., Philip, P., and Bioulac, B. (1999). Morningness/eveningness and the need for sleep. *Journal of Sleep Research*, 8(4):291–295.

Tang, J. C., Yankelovich, N., Begole, J., Van Kleek, M., Li, F., and Bhalodia, J. (2001). Connexus to awarenex: extending awareness to mobile users. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 221–228. ACM.

Tang, J. C., Zhao, C., Cao, X., and Inkpen, K. (2011). Your time zone or mine?: a study of globally time zone-shifted collaboration. In *Proceedings of the ACM 2011 conference on Computer supported cooperative work*, pages 235–244. ACM.

Tani, T. and Yamada, S. (2013). Estimating user interruptibility by measuring table-top pressure. In *CHI'13 Extended Abstracts on Human Factors in Computing Systems*, pages 1707–1712. ACM.

TimeDoctor (2019). `https://timedoctor.com`. Retrieved January 28, 2019.

Tims, M., Bakker, A. B., and Derks, D. (2012). Development and validation of the job crafting scale. *Journal of Vocational Behavior*, 80(1):173 – 186.

Toscos, T., Faber, A., An, S., and Gandhi, M. P. (2006). Chick Clique : Persuasive Technology to Motivate Teenage Girls to Exercise. In *CHI '06: CHI '06 extended abstracts on Human factors in computing systems*, pages 1873–1878.

Tourangeau, R., Rips, L. J., and Rasinski, K. (2000). *The psychology of survey response*. Cambridge University Press.

Travers, C. J., Morisano, D., and Locke, E. A. (2015). Self-reflection, growth goals, and academic outcomes: A qualitative study. *British Journal of Educational Psychology*, 85(2):224–241.

Treude, C., Filho, F. F., and Kulesza, U. (2015). Summarizing and Measuring Development Activity. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 625–636.

Treude, C. and Storey, M.-A. (2010). Awareness 2.0: staying aware of projects, developers and tasks using dashboards and feeds. In *Software Engineering, 2010 ACM/IEEE 32nd International Conference on*, volume 1, pages 365–374. IEEE.

Tulving, E. and Pearlstone, Z. (1966). Availability versus accessibility of information in memory for words. *Journal of Verbal Learning and Verbal Behavior*, 5(4):381–391.

Tuten, T. L. and Bosnjak, M. (2001). Understanding differences in web usage: The role of need for cognition and the five factor model of personality. *Social Behavior and Personality: an international journal*, 29(4):391–398.

Van Der Voordt, T. J. (2004). Productivity and employee satisfaction in flexible workplaces. *Journal of Corporate Real Estate*, 6(2):133–148.

Van Solingen, R., Berghout, E., and Van Latum, F. (1998). Interrupts: just a minute never is. *IEEE software*, (5):97–103.

Vasilescu, B., Blincoe, K., Xuan, Q., Casalnuovo, C., Damian, D., Devanbu, P., and Filkov, V. (2016a). The sky is not the limit: Multitasking on GitHub projects. In *International Conference on Software Engineering*, ICSE, pages 994–1005. ACM.

Vasilescu, B., Blincoe, K., Xuan, Q., Casalnuovo, C., Damian, D., Devanbu, P., and Filkov, V. (2016b). The Sky is Not the Limit: Multitasking on GitHub Projects. pages 994–1005.

Wagner, S. and Ruhe, M. (2008). A Systematic Review of Productivity Factors in Software Development. In *Software Productivity Analysis and Cost Estimation (SPACE 2008)*, pages 1–6.

Wakatime (2019). `http://wakatime.com`. Retrieved March 19, 2019.

Walston, C. E. and Felix, C. P. (1977). A method of programming measurement and estimation. *IBM Systems Journal*, 16(1):54–73.

Whiteside, S. P. and Lynam, D. R. (2001). The five factor model and impulsivity: Using a structural model of personality to understand impulsivity. *Personality and Individual Differences*, 30(4):669–689.

Whittaker, S., Hollis, V., and Guydish, A. (2016). Don't Waste My Time: Use of Time Information Improves Focus. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*.

Whittaker, S. and Schwarz, H. (1999). Meetings of the board: The impact of scheduling medium on long term group coordination in software development. *Computer Supported Cooperative Work (CSCW)*, 8(3):175–205.

Williams, A. C., Kaur, H., Mark, G., Thompson, A. L., Iqbal, S. T., and Teevan, J. (2018). Supporting workplace detachment and reattachment with conversational intelligence. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, page 88. ACM.

Wiseman, R. (2007). *Quirkology: How we discover the big truths in small things*. Basic Books.

Wood, J. V. (1989). Theory and research concerning social comparisons of personal attributes. *Psychological Bulletin*, 106(2):231–248.

Wu, I.-C., Liu, D.-R., and Chen, W.-H. (2005). Task-stage knowledge support: coupling user information needs with stage identification. In *IRI -2005 IEEE International Conference on Information Reuse and Integration, Conf, 2005.*, pages 19–24.

Xia, X., Bao, L., Lo, D., Xing, Z., E. Hassan, A., and Li, S. (2017). Measuring Program Comprehension: A Large-Scale Field Study with Professionals. *IEEE Transactions on Software Engineering*, pages 1–26.

Zhou, M. and Mockus, A. (2010). Developer fluency: Achieving true mastery in software projects. In *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, pages 137–146. ACM.

Zimmerman, B. J. (2006). Development and adaptation of expertise: The role of self-regulatory processes and beliefs.

Zou, L. and Godfrey, M. W. (2012). An industrial case study of coman's automated task detection algorithm: What worked, what didn't, and why. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pages 6–14.

Züger, M., Corley, C., Meyer, A. N., Li, B., Fritz, T., Shepherd, D., Augustine, V., Francis, P., Kraft, N., and Snipes, W. (2017). Reducing Interruptions at Work: A Large-Scale Field Study of FlowLight. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*, pages 61–72.

Züger, M. and Fritz, T. (2015). Interruptibility of software developers and its prediction using psycho-physiological sensors. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 2981–2990. ACM.

Züger, M., Müller, S. C., Meyer, A. N., and Fritz, T. (2018). Sensing interruptibility in the office: A field study on the use of biometric and computer interaction sensors. In *CHI*.

Curriculum Vitae

Personal Information

Name	André N. Meyer
Nationality	Swiss
Date of Birth	May, 9, 1991
Place of Birth	Uster, Switzerland

Education

2015 – 2019	Doctoral program in Informatics Department of Informatics University of Zurich, Switzerland
2013 – 2015	Master of Science in Informatics Department of Informatics University of Zurich, Switzerland
2009 – 2013	Bachelor of Science in Informatics Department of Informatics University of Zurich, Switzerland